'68' MICRO JOURNAL

$2.95 USA

Australia	A $ 4.75	New Zealand	NZ $ 6.50
Singapore	S $ 9.45	Hong Kong	H $23.50
Malaysia	M $ 9.45	Sweden	30:-SEK

VOLUME V ISSUE XII • Devoted to the 68XX User • December 1983

"Small Computers Doing Big Things"

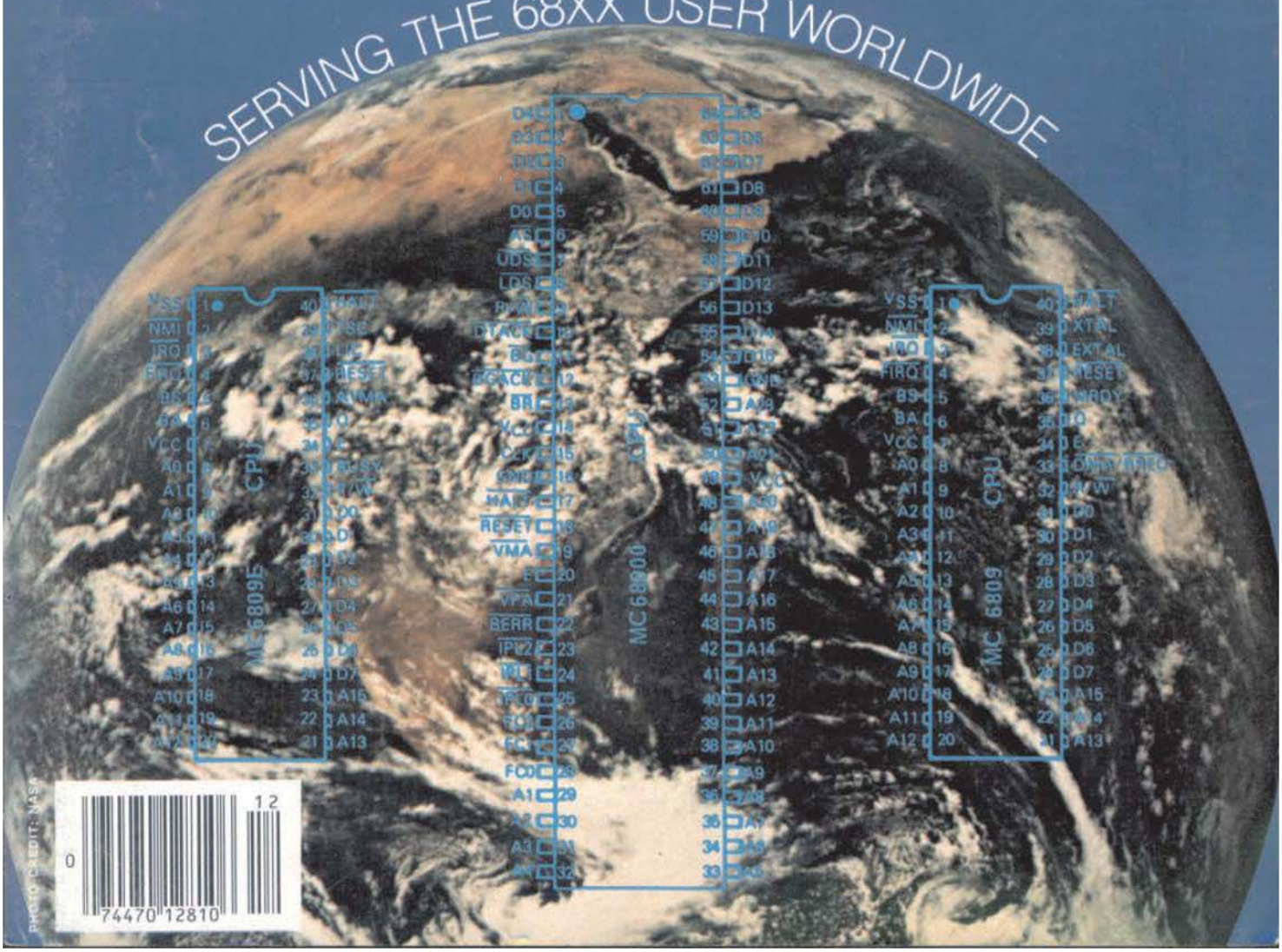


PHOTO CREDIT: NASA

0 74470 12810 12

FREE FROM LIMITATIONS, A TEACHER CAN UNLOCK IMAGINATIONS.

By 1985, fully 75% of all jobs will involve computers. So for today's teachers, selecting the right computer is an awesome responsibility. It's a decision you can feel confident about with Southwest Technical Products Corporation's S + computer system. The S + is specially designed for a student's tomorrow and a teacher's today. It's built tough. It's simple to operate. You can teach without becoming a hardware specialist.

The S + is fluent in a variety of languages: BASIC, PASCAL, COBOL, FORTRAN, PILOT, FORTH, "C," and MUMPS and allows you simultaneous use in the classroom. This flexibility enables you to use your own curriculum, not a machine's version. The system also allows for individual learning styles. Students go as fast and as far as the teacher chooses.

Even though it's priced like a smaller personal computer, the S + has the multi-task power and expansion abilities of a professional computer. It can grow with you and your students.

Tomorrow is in your hands, today. Call Fisher Scientific, exclusive distributor of our S + system, and you'll get the key to unlock imaginations.

For more information.
Call toll-free 800-621-4769.
Illinois customers call collect (312) 378-7770.

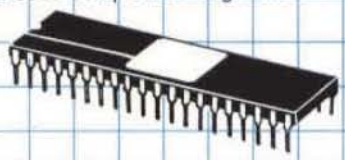
Fisher Scientific
An ALLIED Company

Educational Materials Division
4901 W. LeMoyne St.,
Chicago, Illinois 60651

MICROWARE'S OS-9 IS NUMBER ONE.

OS-9 NOW HAS THE LARGEST USER COMMUNITY

More users now run OS-9 on their 6809 computers than all other operating systems combined. This outstanding success story was no accident — it's due to OS-9's technical excellence backed up by outstanding Microware support. OS-9's Unix-type architecture and totally modular design gives your computer more power and versatility. OS-9 also gives you more possibilities for customization so you can tailor your system exactly to your needs. And aren't flexibility and performance the reasons you chose a 6809 computer to begin with?



OS-9 HAS BEEN CHOSEN BY OVER 50 6809 SYSTEM MANUFACTURERS

OS-9 is now offered as a standard operating system by almost every 6809 system manufacturer, and has been designed into an amazing variety of dedicated systems and products including personal and business computers, process control systems, data and telecommunications systems, and more. In all, over 50 companies and organizations have obtained OS-9 distribution licenses including such well-known names such as General Motors, NASA, Fujitsu, Western Electric, Motorola, Sykes Datatronics, Eastman Kodak, Thomson-CSF, and Tandy Corp.

OS-9 GIVES YOU A SOFTWARE BASE TO BUILD ON

Whatever your application, OS-9 speaks your language! Microware offers BASIC09, an Extended/ Structured Basic, a complete C Compiler, a full ISO Pascal Compiler, the ANSI Standard CIS Cobol Compiler, plus Relocatable Macro Assembler. These high performance programming languages are all fully implemented and deliver unmatched performance and outstanding features. Additionally, OS-9 compatible applications packages such as word processors, screen editors, spreadsheets, business software, and utilities are offered by a rapidly growing number of third-party software houses.

PLUS OUTSTANDING MICROWARE SUPPORT: WE KEEP IN TOUCH WITH YOU

Even when you have the best software and documentation, there can be times when you need questions answered. That's why Microware is committed to giving OS-9 users the best possible **personalized** service. Here are some of the ways we deliver solid support:

- A Software Support Hotline for direct access to our technical staff
- "Pipelines", our free quarterly newsletter
- OS-9 User Seminars, the annual OS-9 community gathering
- a liberal update policy for new releases

Microware does business on a person-to-person basis. When you call you'll find yourself speaking with someone who's both knowledgeable and genuinely interested in helping.

YOU CAN COUNT ON OS-9 NOW AND IN THE FUTURE

Microware is not standing still — we're firmly committed to continuing support for the 6809 and we will continue to introduce exciting new software products for the 6809. We will soon announce OS-9/68000 and programming languages for the 68000 which will be upward compatible with 6809 versions, so if and when you are ready for the 68000 your OS-9 software can go with you.

MICROWARE.

Microware Systems Corporation
Box 4865 • Des Moines, IA 50304
515-279-8844 Telex 910-520-2535

OS-9: BETTER BY DESIGN

'68' MICRO JOURNAL

Portions of the text for 68 MICRO JOURNAL was prepared using the following furnished hard/software.

COMPUTERS-HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, Texas 78216
S09-5/8 DMF disk-COS1-8212W-Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe-OS9-FLEX-Assorted Hardware

EDITORS-WORD PROCESSORS
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX-Editor-Processor

Great Plains Computer C mpany, Inc.
PO Box 916
Idaho Falls, ID 83401
STYLO-Mail Merge

CONTENTS

Vol. V, Issue XII DECEMBER '83

Send All Correspondence To:

Computer Publishing Center
68 MICRO JOURNAL
5900 Cassandra Smith
PO Box 849
Hixson, TN 37343
615 842-4600

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tenn. and additional entries. Postmaster: send Form 3579 to 68' Micro Journal, PO Box 849, Hixson, Tennessee.

SUBSCRIPTION RATES
USA
1-Year $24.50 2-Years $42.50 3-Years $64.50
FOREIGN

See Page 52

Items Submitted for Publication

Articles submitted for publication should be accompanied by the authors **full name, address,** date and **telephone number.** It is preferred that articles be submitted on either 5 or 8 inch diskette in TSC Editor format or STYLO format. All diskettes will be returned.

The following TSC Text Processor commands **ONLY** should be used (due to our proportional processor): .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. The rest we will enter at time of editing.

STYLO commands are all acceptable except the ,pg page command, we print edited text files in continous text.

All articles submitted on diskettes should be in TSC FLEX™ format, either FLEX2 6800, or FLEX9 6809 any version.

If articles are submitted on paper they should be on white 8X11 bond or better grade paper. No hand written articles (hand written or drawn art accepted). All paper submitted articles will be photo reproduced. This requires that they be typed or produced with a dark ribbon (no blue), single spaced and type font no smaller than 'elite' or 12 pitch. Typed text should be approximately 7 inches wide (will be reduced to column width of 3 1/2 inches). **Please use a dark ribbon!**

All letters to the editor should also comply with the above and bear a signature. Letters of 'gripes' as well as 'praise' are solicited. We attempt to publish all letters to the editor verbatim, however, we reserve the right to reject any submission for lack of 'good taste'. We reserve the right to define what constitutes 'good taste'.

Advertising: Commercial advertisers please contact the 68 Micro Journal advertising department for current rate sheet and requirements.

Classified: All classified must be non-commercial. Maximum 20 wo ds per classified ad. Those consisting of more than 20 words should be figured at .35 cents per word. 20 words or less $7.50 minimum, one time, paid in advance. No classified ads accepted by telephone.

TO REALIZE THE FULL POWER & PERFORMANCE OF THE 6809, LOOK TO GIMIX.

GIMIX OFFERS YOU A VARIETY OF SS50 BUS COMPONENTS AND SYSTEMS.

OS-9 GMX III

The GMX 6809 CPU III and OS-9 GMX III. A Multi-user, Multi-tasking package for the ultimate in System Performance plus protection of the system and other users from crashes caused by errors in individual users programs.

#01 (CPU & Software) ..... NEW! **$1698.01**

INTELLIGENT I/O PROCESSOR BOARDS increase system throughput by reducing interrupts to the host, buffering data transfers, and data preprocessing. Prices include on board firmware. Requires system drivers.

#11 3 port RS232 Serial (SS30) ..... NEW! **$498.11**
#12 4 port Parallel (SS50) **$598.12**
OS-9 GMXIII drivers.. (included when purchased with GMX III package) .. **$200.00**
OS-9 Level 2 users -- contact GIMIX for system requirements and availability.

192K GMX III #79 SYSTEMS: All include GMX 6809 CPU III and OS-9 GMX III (#01); a #11 3 port Intelligent serial I/O & cables; #19 Classy Chassis; 192KB Static RAM; #68 DMA controller, all necessary cables, power regulators, and filler plates. The OS-9 Editor, Assembler, Debugger, BASIC-09, and RUNB are included.

#79 with dual 40 track DSDD drives **$5998.79**
#79 with dual 80 track DSDD drives ..... NEW! **$6298.79**
#79 with #88 8" Dual Drive Disk System **$7598.79**
#79 with #90 19MB Winchester subsystem & one 80 track DSDD drive .. **$8998.79**
UniFLEX for the GMX 6809 CPU III and Intelligent I/O boards is in development.

OS-9 GMX I; OS-9 GMX II; FLEX; and UniFLEX

The #05 GIMIX 6809 PLUS CPU board **$578.05**
Options: GMX DAT ..... $35.00 SWTPC DAT $15.00
9511A ..... $312.00 9512 ..... $265.00

#49 64KB GHOST SYSTEM includes: #05 CPU; #19 Classy Chassis; 64KB static RAM; a #43 2 port serial card & cables; #68 DMA Controller; all necessary cables, power regulators, and filler plates; GMXBUG monitor; FLEX; and OS-9 GMX I. You can software select either FLEX or OS-9. The OS-9 Editor, Assembler, Debugger, BASIC-09, and RUNB are also included.

#49 with dual 40 track DSDD drives **$4398.49**
#49 with dual 80 track DSDD drives **$4698.49**
#49 with #88 8" Dual Drive Disk System **$5998.49**
#49 with #90 19MB Winchester subsystems & one 80 track DSDD drive **$7398.49**

#39 128KB SYSTEM includes: #05 CPU w/DAT; #19 Classy Chassis; 128KB of static RAM; a #43 2 port serial card & cables; #68 DMA Controller; all necessary cables, power regulators, and filler plates; GMXBUG monitor; FLEX; and OS-9 GMX II. You can software select either FLEX or OS-9. The OS-9 Editor, Assembler, Debugger, BASIC-09, and RUNB, and GMX-VDISK for FLEX are included.

#39 with dual 40 track DSDD drives **$4998.39**
#39 with dual 80 track DSDD drives **$5298.39**
#39 with #88 8" Dual Drive Disk System **$6598.39**
#39 with #90 19MB Winchester subsystem & one 80 track DSDD drive **$7998.39**
UniFLEX, available at extra cost, requires 8" or Winchester drives. A signed license agreement with TSC is required before shipment.

You can add to any GIMIX system RAM, I/Os and other options, or substitute non-volatile RAM. GIMIX will customize to your needs.

COMING SOON: Contact GIMIX for price and availability on 40MB and 72MB Winchester (5¼") drives, removeable pack Winchesters, 256KB static RAM boards.

All GIMIX systems are guaranteed for 2MHz operation. GIMIX systems include documentation for all boards and software in a GIMIX binder. **ALL DRIVES ARE 100% TESTED AND ALIGNED BY GIMIX.**

ALL BOARDS AND SYSTEMS ARE ASSEMBLED, BURNED-IN, AND TESTED. GOLD-PLATED BUS CONNECTORS ARE USED.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account #73-32033. Visa or Master Charge also accepted.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.

GIMIX Inc. reserves the right to change pricing, terms, and product specifications at any time without further notice.

ALL PRICES ARE F.O.B. CHICAGO

Choose from GIMIX' wide variety of system components.

The GIMIX CLASSY CHASSIS #19 consists of a heavyweight aluminum cabinet, constant voltage ferro-resonant power supply, and SS50 Mother board with baud rate generator board $1398.19

Triple Disk regulator card and cables $88.22
Missing cycle detector $38.23
Back panel connector plates (specify) $8.60
Baud rate generator card $88.93
Filler plates $14.92
50 Hz. option $30.00

MEMORIES (GIMIX uses only Static RAM)

#67 64KB NMOS STATIC RAM board **$478.67**
#64 64KB CMOS STATIC RAM board w/battery back-up **$568.64**
#34 8K PROM board **$98.34**
#32 16 socket PROM/ROM/RAM board **$238.32**

I/O Boards (see above for Intelligent I/Os)

#41 Single port serial, RS232/20ma. current loop **$88.41**
#43 2 port serial, RS232 ..... **$128.43**
#46 8 port serial, RS232 ..... **$318.46**
#42 2 port parallel **$88.42**
#45 8 port parallel **$198.45**
#50 serial, RS232, RS422, RS423 ..... **$244.50**
#52 SSDA serial, RS232, RS422, RS423 ..... **$254.52**
#54 ADLC serial, RS232, RS422, RS423 ..... **$268.54**
Each cable with connectors for back panel mounting (specify board) **$24.95**

DISK CONTROLLERS

#68 DMA (featured in all systems above) **$568.68**
#28 dbl. dens. programmed I/O (5" drives only) **$298.28**
#58 single dens. programmed I/O (5" and/or 8" drives) **$226.58**
#48 same as **#58** but for 5" drives only **$188.48**
Cable sets: 8" with Back Panel connector **$29.25**
for two 8" external drives **$44.26**
for two 5" drives **$34.96**

SOFTWARE: GIMIX exclusive versions of OS-9/GMX I, II, III & FLEX are for GIMIX hardware only. All versions of OS-9 require the #68 controller.
When ordered with any controller, FLEX is **$30.00**

GMXBUG PROMs and manual **$98.65**
Boot or Video boot PROM ..... **$30.00** UniFLEX boot PROM **$58.00**
OS-9 GMX I ..... **$200.00** OS-9 GMX II **$500.00**
Editor ..... **$125.00** Assembler **$125.00**
BASIC-09 ..... **$200.00** RUNB **$100.00**

DISK DRIVES FOR GIMIX SYSTEMS – complete with cables and power regulators.

5" DSDD 40 track ..... 2 for **$900.00**
5" DSDD 80 track ..... 2 for **$1300.00**
#88 8" Dual 8" DSDD drives, cabinet, power supply, & cables **$2698.88**
Cabinet only ..... **$848.18** 220V 50Hz. Option, add **$30.00**
Filler plate ..... **$14.83** Cable for 2 drives **$44.82**
Cable for 4 drives ..... **$67.84** Cable for cabinet to mainframe **$45.81**

WINCHESTER SUBSYSTEMS: for use only in GIMIX systems with **#68** DMA controller.

#90: Includes one 19MB drive, interface, and Software **$3589.90**
#91: includes two 19MB drives, interface and Software **$5288.91**
Contact GIMIX for price and availability of other forthcoming subsystems.

OTHER BOARDS

#76 GHOST 80X24 VIDEO BOARD **$398.76**
#56 50 pin Proto-boards ..... **$56.66** **#33** 30 pin Protoboards **$38.33**
#03 6800 CPU **$224.03**
#08 6800 CPU with timers ..... **$288.06** Baud rate option, add **$30.00**

#08 RELAY DRIVER (board, bracket, transformer, and 31 relays) **$1128.08**
#86 - #08 (board, bracket, transformer, without relays) **$538.88**
#85 OPTO board **$348.85**
WINDRUSH EPROM PROGRAMMER **$375.00**
3" Binder ..... 12.00 2" Binder **$9.00**

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

DON'T SEE IT??? ASK! OUR BROCHURE HAS MORE COMPLETE DESCRIPTIONS AND SPECS. PHONE OR WRITE TODAY FOR YOUR COPY.

BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA Inc. FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc. GIMIX, GHOST, GMX, CLASSY CHASSIS, are registered trademarks of GIMIX, Inc.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

Microware presents 4 new OS-9 software packages.

1 LEVEL II PRINT SPOOLING SYSTEM

This versatile package gives your OS-9 Level Two System a complete print spooling management capability for time-sharing applications. Features of the spooling system are:

- *Handles up to seven independent spooling devices and queues with "print on first available device" feature.*
- *Prints large block header pages between listings with date, time, user name and job name.*
- *Multiple listing copy option.*
- *Complete forms change capability for each job and device.*
- *Prints formatted or unformatted listings.*
- *Status command displays print queues and status.*
- *User can kill or change priority of queued jobs.*

*Available **only** for OS-9 Level Two Systems.*

Suggested List Price: $150.00 Manual Only: $15.00

2 RMA RELOCATABLE MACRO ASSEMBLER

At last — a full feature relocatable macro assembler and linkage editor for OS-9. RMA permits sections of assembly language programs to be independently assembled to "relocatable object files". The linkage editor takes any number of program sections and/or library sections and combines them into a single executable OS-9 memory module. Global data (including indexed and direct addressing modes) and program references are automatically resolved in the process. The macro facility permits commonly used statement sequences to be defined, then used within the program with appropriate parameter substitution. RMA also supports conditional assembly and library source files.

Suggested List Price: $200.00 Manual Only: $20.00

3 OS-9 FILE HANDLER TOOLBOX

Introducing a special toolbox for OS-9 users who do a lot of file manipulation! A collection of 12 useful OS-9 command programs: Most can be used as "filters" using OS-9 pipeline facilities. Included are:

D *— unformatted directory listing with "wild card" matching*
Compress *— does character compression on text files.*
Expand *— restores a "compressed" file to the original state.*
Split *— breaks a file into smaller files.*
Space *— indents lines with optional spacing between lines.*
Code *— decodes any key on a keyboard to hex.*
Qsort *— quick sort for small files, directories, etc.*
Pr *— versatile formatted file printing utility.*
Tr *— transliterates text pattern to substitution pattern.*
Grep *— searches file for a pattern and prints matching lines.*
Xmode *— same "tmode" except changes are made to the device descriptor.*
Count *— counts words, lines, or characters within a text file.*

Suggested List Price $85.00

4 ENTERTAINMENT PACK I

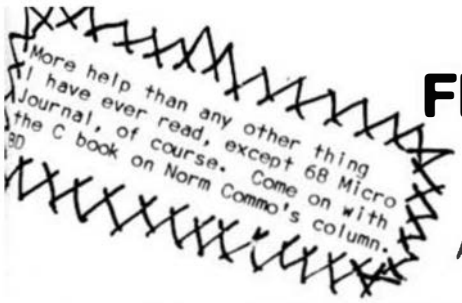
A collection of games and other interesting programs that are not only entertaining but serve as good instructional examples of Basic09 programming techniques. All programs include complete Basic09 source files and can be easily edited to run on standard alphanumeric or graphics terminals.

Blkjak *— A Vegas-rules blackjack game.*
Clk *— graphical display of a wall clock on your terminal.*
Dogs *— Greyhound racing with simulated graphics.*
Eliza *— Basic09 version of the famous artificial intelligence simulation of natural language dialogue with a psychiatrist.*
Haiku *— Program that creates original "haiku" prose.*
Quest *— a mini-"Adventure" game.*
Rats *— find your way out of a computer-generated maze — from a rat's point of view.*
Towers *— a graphical display of the solution to the "Tower of Hanoi" puzzle.*

Suggested List Price: $85.00

MICROWARE®

Microware Systems Corporation
P.O. Box 4865 • Des Moines, IA 50304
515-279-8844 • Telex 910-520-2535



FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES,** in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

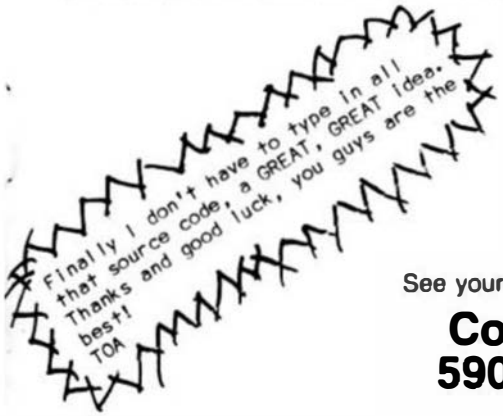
All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1, .C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.



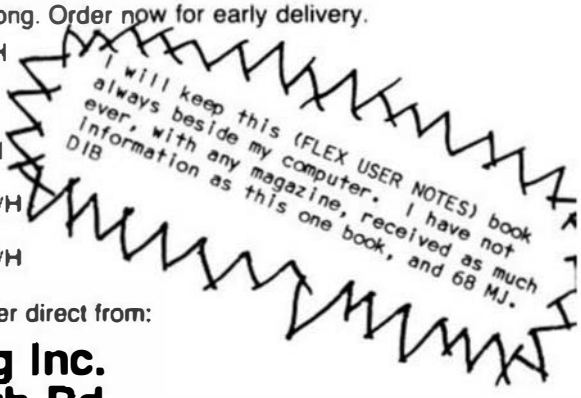
Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

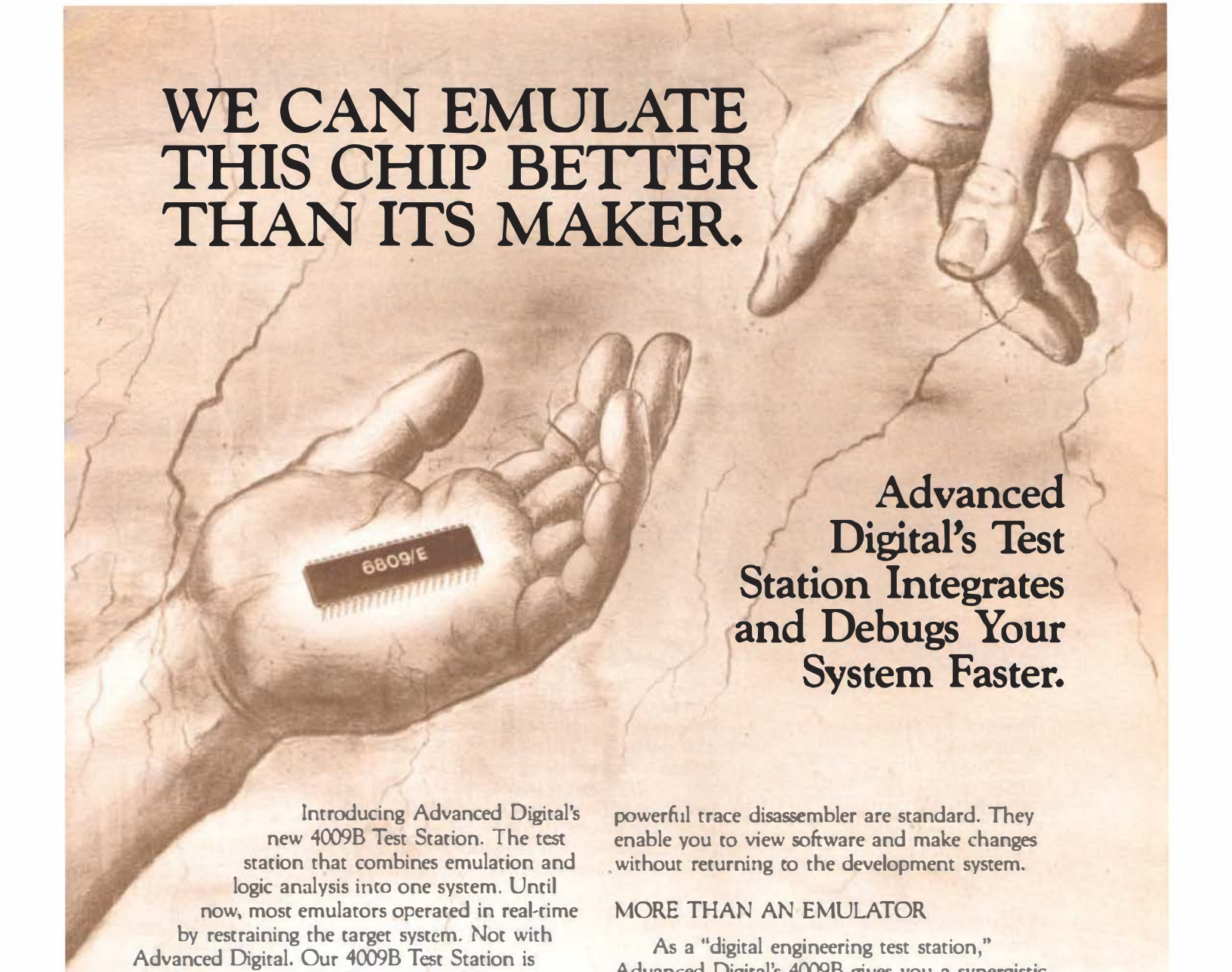


See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

™FLEX is a trademark of Technical Systems Consultants

WE CAN EMULATE THIS CHIP BETTER THAN ITS MAKER.



Advanced Digital's Test Station Integrates and Debugs Your System Faster.

Introducing Advanced Digital's new 4009B Test Station. The test station that combines emulation and logic analysis into one system. Until now, most emulators operated in real-time by restraining the target system. Not with Advanced Digital. Our 4009B Test Station is totally transparent. Totally real-time. We meet or exceed manufacturer's timing specifications. This is particularly important when emulating interrupt intensive systems and accommodating DMA operation.

Universal operation enables you to emulate 6809 and 6809E series microprocessors without making software or hardware changes. Up to 128K of memory overlay, four hardware triggers and 2K deep of trace help you pinpoint problems faster. A line-by-line assembler and a powerful trace disassembler are standard. They enable you to view software and make changes without returning to the development system.

4009B TEST STATION

MORE THAN AN EMULATOR

As a "digital engineering test station," Advanced Digital's 4009B gives you a synergistic approach to integration problems. Two micto-processors control the system. One is dedicated to emulation while the second microprocessor supervises the 32 channel logic analyzer and all system menus. This design lets you emulate and perform all system functions simultaneously.

Discover the most powerful diagnostic tool available for hardware and software integration.

Call Advanced Digital Technology for a free demonstration.

Advanced Digital Technology

13400 Northup Way, #27
Bellevue, WA 98005
(206) 643-2382

TMP (Total Management Planning) OS-9 SOFTWARE

THESE OFF-THE-SHELF APPLICATION PACKAGES HAVE CHANGED THE PICTURE FOR PROFESSIONAL SOFTWARE USERS

Here are the comprehensively supported OS-9 application software packages that discerning professionals are selecting today. Each one runs stand-alone or in combination with other TMP packages.

TMP/CALC is a *new-generation* "Electronic Spreadsheet" package that goes a quantum leap beyond ordinary Calc software. Superior speed, and extra features like Dynamic Calculations and Dynamic Overlay, mean you can change a cell *anywhere* on the spreadsheet and get automatic updates wherever they belong. No more "wrap-around" since output to the printer is formatted. "Help" screens prompt without erasing data, and the rows and columns on any worksheet are limited *only* by available memory.
ONLY $250

TMP/Manager is a structured database manager, and is the ultimate driving force behind any total *Management/Marketing Planning* system. Its built-in select/sort/merge module — and optimized machine code — makes it superfast, and provides unusually nimble data manipulation characteristics. Other outstanding features are detailed prompting, easy identification of data fields, and an excellent security system.
ONLY $500

TMP FreeForm fills the void between a structured DBMS and word processing; it's an "electronic index card" package with an endless list of valuable applications such as parts cataloging, look-up inventories and listings of *any* kind. Type anything on up to 32,000 electronic "cards" of up to 9 pages each that reside in "file drawers" (databases). Up to 13 key-words can call each page.
ONLY $150

TMP/Front-End integrates any or all of the packages described above, and integrates TMP with word-processors, BASIC and other high-level languages.
NO CHARGE with TMP/Manager

TMP T.M.

TMP PACKAGES ARE EXCEPTIONALLY FRIENDLY. EACH COMES WITH MULTI-MEDIA TRAINING AND OUTSTANDING DOCUMENTATION

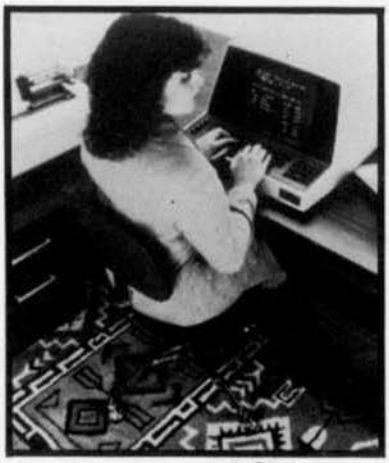
AUDIO CASSETTE TRAINING TAPES are provided with each TMP software package. These crisp, professionally produced training aids take a new user through "hands-on" exercises that instantly build confidence, and minimize reference to manuals.

CLEAR DOCUMENTATION is specially prepared for each TMP package. Remarkably readable because they are conversational in style and logically organized, this series of manuals *further* assures quick user comfort and productivity with TMP software. Includes sections on how to best utilize the system; integrating user-written software with TMP packages, and other bonus topics.

VIDEOTAPE PROGRAMS are available to TMP Dealers. They dramatically take the viewer into real business environments for a close look at how a TMP software package is utilized. Video programs can be personalized for specific organizations. Programs may be specified for most popular tape formats.

THE BOTTOM-LINE IS THIS: Software is only as effective as its training and documentation. TMP packages are *incomparable* in both areas.

RUN TMP IN A SMOKE SIGNAL CHIEFTAIN™ COMPUTER TO EXPERIENCE THE KIND OF SPEED AND POWER YOU CAN EASILY AFFORD *TODAY*



Endurance-Certified Chieftains are consistently among the fastest computers in CPU and I/O speeds according to the widely respected BENCHMARK REPORTS. That, combined with almost legendary reliability, makes Chieftains the logical choice in computers using OS-9 operating systems.

Configurations range from floppy-disk systems to multi-user, multi-tasking Winchester hard disk systems with tape back-up; performance approaching mainframes at prices you can live with.

TMP software packages for most OS-9 systems and Chieftain computers are offered exclusively from Smoke Signal. Inquiries from non-Smoke Signal dealers are invited. Call (213) 889-9340.

SMOKE SIGNAL
Chieftain Computers
31336 Via Colinas • Westlake Village, CA 91362
(213) 889-9340

Flex User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

Taming the CT-82XX

If you have a CT-82, CT-8212, or CT-8209 terminal, yo have a device with a nice green screen and lots of smart features. You also have one "feature" that is absolutely useless. There is a key on your terminal called a SHIFT LOCK. Now most terminals have what they call a CAPS LOCK. The CAPS LOCK simply causes all lower case letters to become upper case, so you can emulate an old teletype, or write a program in a language that requires upper case characters without constantly going for the shift key. Of course without caps lock you have to release the shift key for numbers and punctuation, and depress it for letters, and therein lies the problem with a SHIFT LOCK function.

Some of the software suppliers of editors have come to the rescue with a built in software CAPS LOCK. You know, if you have a CT-82XX that Stylo has such a feature. I recently received a copy of Dynastar for test and review, and I noticed that it doesn't have this feature, and so is rather hard to use with a CT-82XX. It occurs to me that there might be some other screen editors without this feature, that might benefit from some help too, so I set out to write an INCHNE (input character with no echo) routine to replace the one in SBUG-E.

It turns out that all the screen oriented editors use INCHNE so that they don't echo the control characters that you must type while editing text. I found that all calls to this routine are done by an indirect jump through the location $D3E5. The listing of CAPS shown here is the result of my effort, and I guarantee it will work with any version of FLEX9 except the very early original FLEX09, which was a reassembly of the old 6800 FLEX, and was superceded within about 6 months. Any version of FLEX newer than about Feb. 1980 will have this jump vector.

What the routine does is to set up a CAPS FLAG, which initially is cleared so the routine doesn't modify characters from the terminal. When the routine is called, it looks for a control underline ($1F) character, and when one is detected it "toggles" the CAPS FLAG. When the CAPS FLAG is ON, all input characters are tested to see if they are within the range of the values of a to z and if they are, they are modified to be A to Z, the upper case versions. The control underline is not passed through to the computer, so that it is lost as a possible control code to the software being run. This character could be changed to be any unused code for special applications. CAPS is assembled and then installed either by use of the GET CAPS.BIN command, or by appending it to the editor or other software so it is loaded with the software.

I decided to get a bit more general about it and to modify the INCH (with echo) routine also. CAPSALL is a listing of that attempt. I found to my chagrin that FLEX doesn't use the INCH vector it has set up (at least in General FLEX), at $D3FB. That ve tor jumps to location $D383 which does an indirect jump to the location whose address is found at $D385, the SBUG-E entry point. Unfortunately in FLEX (GIMIX DMA VERSION) at $CD09, the GETCHR routine, does a jump to $D383, so it is the address at $D385 that must be modified to cause a jump to the substitute INCH. Unfortunately the jump is indirect, so CAPSALL contains a FDB INCH, and the address of that FDB (called INCHV) is what is overlaid at $D385. Your version of FLEX may not be the same, and this more general version that I call CAPSALL may not work without a slight modification.

I did some research on various FLEX versions,and found that the location for the overlay is always within a few bytes of $D380, and you can look with the monitor for the double byte $F804. This is the location that must be overlaid with the INCHV. Incidentally, all the CC FLEX versions are derived from general FLEX, and they apparently do all use the vector at $D3FB. The CC versions, obviously don't have SBUG-E so the eventual jump is NOT to $F804, but to some other location. You would only be interested in patching CC FLEX if you have an external CT-82XX attached to it anyway.

I put the new input routines at $BE00, and fixed memend to point at $BDFF so the routines won't get wiped out by any standard software that honors memend. They seem to work quite well. I can toggle CAPS LOCK off and on even while in the FLEX command mode with the +++ prompt present. FLEX ignores control characters, but all that is passed to it is a null ($00) just to be safe. Operation of Dynastar went from nearly impossible for writing Pascal programs to being just like the operation with a different terminal or with Stylo. Stylo uses the control up-arrow to toggle its caps lock, but that conflicts with a necessary control character in another editor that I have, so I chose control underline.

It occurs to me that this sort of an input "filter" could be extended to detect other control characters and a dumb terminal could be made to look rather smart by putting the software in the computer rather than the terminal. I'd be interested to hear about anyone carrying this idea further.

68000 News

I've received a copy of a "press release" from TSC indicating that they have what looks like our familiar Extended BASIC with some enhancements, available in a 68000 version to run under the 68000 UNIX operating systems. They also announce a version of their pre-compiler for this BASIC. The description sounds just like 6809 Extended BASIC complete with sequential and random files, and virtual arrays. They have the error trapping facilities, ERR and ERL to report error numbers and lines, and the 16.8 digit arithmetic. I don't want to imply that this is a "straight" translation. There are many enhancements such as an "approximately equal" test for floating point numbers.

I recently mentioned being asked if anyone was doing something about either Hardware or Software for the 68000. I can tell you that I have it on very good authority that GIMIX is working on a SS-50 bus version of a 68000 based system. I don't have a completion date to report, but it could well be before you read this.

EDITORIAL

Here we go again. I've been looking at the computer book se tion of a couple of bookstores, local and not so local, and I have some observations. There are quite a few feet of shelves full of "computer b s", among which are quite a few well written books on such subjects as Pascal or "C" programming. These, however are getting to be hard to find among the titles such as "Wordstar Simplified" "Wordstar made easy" "Understanding CP/M", "Programs in Color Computer Extended BASIC" "How to use Visicalc", and on and on seemingly endless y with books EXPLAINING WHAT SHOULD BE COVERED THOROUGHLY AND UNDERSTANDABLY IN THE MA UALS FOR THE VARIOUS SOFTWARE PRODUCTS.

The availability of so many books is a sad commentary on the "understandability factor" of much of the documentation that comes with software that is available today. These books (not necessarily actual titles) all sell for $15 to $20. A careful search in the local Dalton Booksellers, showed not a single book on "C" programming. There were Pascal programming b s in abundance, perhaps reflecting the newness of "C". These comments apply to MUCH of what is available today, not ALL. I think in general we SS-50 bus users are lucky. Our software suppliers seem to be doing a better job than software suppliers in general.

What I was looking for, was a good textbook on computer theory. I found one in the area of "Interactive Compiler and Interpreter Design". Close to what I was looking for... How much? $48!!! It seemed that all the textbook variety (hard cover only) ran from $40 to $50! While the Intera tive Compiler Design was not quite what I was looking for, I'd be willing to part with $20 for it, but $48 seems a bit steep.

I can see it coming now. Thirty-five publishers will be bending my ear about the high ost of producing a book and how such a book has a limited market so that they have to charge that much in order to break even. It seems to me that in the age of computers, if a publisher feels the market will be limited, he doesn't need to publish a book in hardcover and typeset. What's wrong with a photo reproduced daisy wheel printout in softcover to keep the costs down? a b of this type is primarily for reference and study, and not for "show". It doesn't have to look pretty on the shelf in my library. Still, that book did say something about books on compiler design not having to be unintelligible to

someone without a degree in computer science.... Maybe I can afford it after all. Better yet, maybe I can get the company to buy it! (It did.)

WHY ME

I seem to have some strange sort of knack for finding bugs in software. I don't necessarily mean for fixing bugs, but for running across them. Way back when FLEX2 came along, TSC, after some delay, finally supplied a rework of Uiterwyk's BASIC for it. Of course I had a "backlog" of old programs in BASIC, and I tried running them. About the third program, I had a problem. After some experimenting I found that the RESTORE statement didn't work with a DATA statement. Perhaps someone thought that with disk files no one would use DATA statements any longer. Of course my old game program had one, and it bombed. TSC was quick to provide a new copy without that problem, as have most all of the software suppliers in general.

Somewhere around what must be pushing a year ago now, I received a new compiler to evaluate. To the chagrin of the supplier (a VERY conscientious one), I continued to find peculiarities for several months and versions as I got into applications and translating some older programs from Pascal. The supplier indicated that he had been selling the product for over a year, and that I was the ONLY user reporting bugs. We wondered together whether the others just didn't happen to be using the features that had the bugs, or didn't bother reporting them. I (and all the other owners of the compiler) received very prompt patches where possible and new versions when a patch wasn't feasible. A couple months ago (about 6 by the time you read this) I received a "final release" version with a very large and well done manual. I have found no bugs since. The supplier was Windrush, and the product PL9.

I could mention several other cases of immediate finding of bugs in new software. In nearly every case, the response from the supplier has been one of surprise and immediate correction. The moral of the story is that if you do find a bug you should report it immediately. You will be HELPING the supplier by pointing out the problem so it can be fixed before he sells another 50 copies and then has to update those additional 50 owners of the software.

I've said it before, but it bears repeating, that you should be sure you have found a bug, and that you are not misinterpreting the instructions. A supplier will pay attention if you can reduce the program that shows up the bug to its simplest form. A test program of a dozen lines that shows up a bug is much more valued by the supplier than 10 pages with a vague description of the problem.

OmegaSoft

In the September '83 issue of this magazine, Don Williams published a letter from me concerning an oversight on my part. I had been rather wrapped up in comparisons of various compilers, and had done a test on several compilers to look at the code they generated for some assignment statements involving simple variables and array elements. OmegaSoft didn't fare so well against a couple of the newer compilers. Bob Reimiller of OmegaSoft phoned me to tell me that I must have used his version 1 of Pascal, and that the newer version was much more efficient in the area of code generation. The comparisons in question were in the July issue of '68'. I've rerun the tests. If you have the July issue handy, refer to page 12. In the second column, just above the result table, I had given the byte count for the compiler in the last and most complex expression. OmegaSoft is shown as generating 62 bytes. Version 2.1 generated 39 in my test, right in there with the best. The table shows OmegaSoft as having generated 376 bytes for the whole set of assignments. The newer version generated 220 bytes, and OmegaSoft Pascal since version 2.0 has had BYTE variables, which were used in the test.

Since that issue I have done some timing comparisons using the BYTE magazine Sieve of Erastosthenes Benchmark program. I had reported OmegaSoft Pascal as running that sieve in 20 seconds on a 2 MHZ system. My tests with version 2.1 indicate the time to be 14 seconds. Bob Reimiller reports results more on the order of 12 seconds. Perhaps some rearrangement of the variable declarations would put the most referenced ones where the referencing instructions would be shorter, and I don't doubt Bob's results. The authors of the article in BYTE did allow taking advantage of specific features of a compiler including order of declaration of variables, declaring them local or global, etc.

Introl

The other day I received a nice letter from Introl Corp. In that July test report, I had mentioned having used Introl "C" integer only version on a friend's system to run the test. I reported that short inte er and integer were the same in Introl "C". I didn't mention that the K&R specification says that is OK. What I didn't see at the time was that I could just as well have used variables of type char for the cases in which bytes would be adequate. I've rerun the tests for that also. The total byte count using integers and chars for the 16 and 8 bit data types respectively, yielded a byte count of 227 rather than the 261 indicated in the July article.

Introl's president also pointed out to me that I had coded the sieve benchmark test algorithm incorrectly in my review of Whimsical in the September issue. One look was all it took to convince me of the truth of that. I will have to plead a "stupidity attack" here, as I did in my reply to Introl. Of course the array must be reinitialized for each of the ten iterations. It was initialized only the first iteration in my coding. I've re-coded the test and rerun it. The results were 11.5 seconds for ten iterations on a 2 MHZ system. The listing is included here. Obviously, I have to retract my statement that these are the fastest results to date with any compiler I have tested. The correct algorithm yields time that puts the execution time in line with PL9's and just below the Windrush, Dyna-C, and Introl "C" compilers.

I'm sorry for the inaccuracies. I've written about the problems with the previous reports, but it seems that I didn't do a whole lot better with my tests. Still, though, I could make the point that we have five compilers that run the Sieve Benchmark in from 10 to 11.5 seconds. Three of those have floating point capability and two have integer only. Three of them are "C" compilers, which must say something for the language and the implementation available.

PROCESSORS

I also received a letter recently from Gary Bergstrom in Ohio. He noted my comments about our 2 MHZ 6809 being compared to a 4 or 6 MHZ Z-80. I'll quote his letter here "Be careful about clock speed discussions. I believe a 6 MHZ Z-80 can use slower memory than a 2 MHZ 6809 (8 MHZ crystal). An 8 MHZ Z-80 requires slightly faster memory than a 2 MHZ 6809. Zilog and Intel talk crystal frequency and Motorola and the 6502 crows talk crystal frequency / 4. Memory access (or bus bandwidth) is a fair comparison (or at least fairer). Compare 4 MHZ Z-80's with 1 MHZ 6809's and 2 MHZ 6809's to 6 or 8 MHZ Z-80's.. and remember, I like the 6809's but fair is fair."

Gary makes a very good point. Anyone else have any comments? I know that the basic "machine cycle" for the 6809 takes place at 1/4 of the crystal frequency. Someone write and tell us if this is also true for the Z-80. Actually since that column, I have reported some more current benchmark timings and added some late compiler implementations, and the results are still very favorable to the 6809.

Well, I'm sorry for the rather fragmented column this time. There were just a lot of little things to report, and I had to give reply time to those who were kind enough to write and set me straight where I had made some errors.

```
{ PRIME BENCHMARK PROGRAM IN WHIMSICAL


BEGIN
   INTEGER SIZE=8190;

   BOOLEAN ARRAY FLAGS [SIZE];
   SMALLINT L;
   INTEGER I, J, K, COUNT;

   FOR L := 0 TO 9 DO
   BEGIN
      FOR I := 0 TO SIZE DO FLAGS[I] := TRUE; { INITIALIZE ARRAY
      COUNT := 0;
      FOR I := 0 TO SIZE DO
      BEGIN
         IF FLAGS [I] THEN
```

```
        BEGIN
          J := I+I+3;
          K := I+J;
          WHILE K <= SIZE DO
          BEGIN
            FLAGS [K] := FALSE;
            K := K+J;
          END;
          COUNT := COUNT+1;
        END;
      END;
      WRITE CHR($0D),CHR($0A);
      WRITE COUNT,' PRIMES ';
    END;
END.


0 ERROR(S) DETECTED
LAST ASSEMBLED ADDRESS: BE24


                    * CAPS LOCK FILTERED INCHNE ROUTINE FOR DYNASTAR
                    * THIS ROUTINE FILTERS THE CHARACTER INPUT STREAM
                    * FOR DYNASTAR, WHICH USES THE INPUT CHARACTER
                    * WITHOUT ECHO ROUTINE OF FLEX, WHOSE ADDRESS IS AT
                    * $D3E5.
                    *
                    * IF THE CAPS FLAG IS NON ZERO, IT SUBTRACTS $20
                    * FROM THE ASCII CHARACTERS IN THE RANGE a-z TO MAKE
                    * THEM A-Z.  IT DOES NOT CHANGE ANY OTHER CHARACTERS SO
                    * IT IS A TRUE CAPS LOCK AND NOT A SHIFT LOCK LIKE THE
                    * KEY ON THE CT-82II TERMINALS.
                    *
                    * THIS PROGRAM IS TO BE ASSEMBLED AND APPENDED TO DS.CMD
                    * SO THAT IT LOADS WITH DS.
                    *
                    * THE CAPS FLAG IS INITIALIZED TO OFF, AND IT IS TOGGLED
                    * WITH A "CONTROL UNDERLINE (^_)" CHARACTER, WHICH THIS
                    * ROUTINE FILTERS OUT SO IT CAN'T BE USED IN A MACRO
                    * DEFINITION WITHIN DYNASTAR.
                    *
               E004 ACIA    EQU   $E004
               CC2B MEMEND  EQU   $CC2B
                    *
                    * ASSUME INITIALIZED BY MONITOR
                    *
CC2B                      ORG   MEMEND    FIX MEMEND
CC2B BDFF                 FDB   $BDFF
                    *
D3E5                      ORG   $D3E5     FIX THE INPUT CHARACTER VECTOR
D3E5 BE00                 FDB   INCHNE
                    *
BE00                      ORG   $BE00
BE00 B6 E004    INCHNE    LDA   ACIA
BE03 44                   LSRA
BE04 24 FA                BCC   INCHNE    WAIT FOR A CHARACTER
BE06 B6 E005              LDA   ACIA+1    GET CHARACTER
BE09 84 7F                ANDA  #$7F      REMOVE PARITY
BE0B 81 1F                CMPA  #$1F      CONTROL UNDERLINE (SAME AS PIE)
BE0D 26 05                BNE   CONTIN    IF NOT TOGGLE CHARACTER
BE0F 73 BE24              COM   CAPS      TOGGLE THE FLAG
BE12 4F                   CLRA            DON'T PASS CONTROL CHARACTER TO EDITOR
BE13 39                   RTS
BE14 7D BE24    CONTIN    TST   CAPS      IS FLAG ON?
BE17 27 0A                BEQ   CONT2     DONE IF IT IS OFF
BE19 81 61                CMPA  #'a       ELSE SEE IF CHARACTER IS LOWER CASE
BE1B 25 06                BLO   CONT2     DONE IF TOO SMALL
BE1D 81 7A                CMPA  #'z
BE1F 22 02                BHI   CONT2     DONE IF TOO LARGE
BE21 84 DF                ANDA  #$DF      ELSE MAKE IT UPPER CASE
BE23 39         CONT2     RTS
BE24 00         CAPS      FCB   0
                          END


                    * CAPS LOCK FILTERED INCHNE AND INCH ROUTINES
                    * THIS ROUTINE FILTERS THE CHARACTER INPUT STREAM
                    * WHICH USES THE INPUT CHARACTER
                    * WITHOUT ECHO ROUTINE OF FLEX, WHOSE ADDRESS IS AT
                    * $D3E5 AND THE INCH WITH ECHO AT $D3F3.
                    *
                    * IF THE CAPS FLAG IS NON ZERO, IT SUBTRACTS $20
                    * FROM THE ASCII CHARACTERS IN THE RANGE a-z TO MAKE
                    * THEM A-Z.  IT DOES NOT CHANGE ANY OTHER CHARACTERS SO
                    * IT IS A TRUE CAPS LOCK AND NOT A SHIFT LOCK LIKE THE
                    * KEY ON THE CT-82II TERMINALS.
                    *
                    * THE CAPS FLAG IS INITIALIZED TO OFF, AND IT IS TOGGLED
                    * WITH A "CONTROL UNDERLINE (^_)" CHARACTER, WHICH THIS
                    * ROUTINE FILTERS OUT SO IT CAN'T BE USED BY SOFTWARE.
                    *
                    *
               E004 ACIA    EQU   $E004
               CC2B MEMEND  EQU   $CC2B
                    *
                    * ASSUME INITIALIZED BY MONITOR
                    *
CC2B                      ORG   MEMEND    FIX MEMEND
CC2B BDFF                 FDB   $BDFF
                    *
D3E5                      ORG   $D3E5     FIX THE INPUT CHARACTER VECTOR
D3E5 BE00                 FDB   INCHNE
                    *
                    * NOTE THE FOLLOWING ORG SHOULD BE SUCH THAT THE FDB
                    * OVERLAYS
                    * THE HEX VALUE F006 IN SOME CODE THAT DOES A JMP(#F006).
                    * $F006 IS THE ADDRESS FOR AN INDIRECT JUMP
                    * TO SBUG-E ROUTINE INCHE, INPUT WITH ECHO.
                    *
                    * THIS LOCATION WAS BEEN FOUND TO VARY BY A FEW BYTES AROUND
                    * THE VICINITY OF $D380.  YOU WILL HAVE TO DO A MEMORY DUMP OF
                    * THAT AREA AND DETERMINE THE ADDRESS WHERE $F006 IS FOUND
                    *
D385                      ORG   $D385     THIS IS WHERE $F006 IS IN MY VERSION OF FLEX
D385 BE25                 FDB   INCHV
                    *
BE00                      ORG   $BE00
BE00 B6 E004    INCHNE    LDA   ACIA
BE03 44                   LSRA
BE04 24 FA                BCC   INCHNE    WAIT FOR A CHARACTER
BE06 B6 E005              LDA   ACIA+1    GET CHARACTER
BE09 84 7F                ANDA  #$7F      REMOVE PARITY
BE0B 81 1F                CMPA  #$1F      CONTROL UNDERLINE (SAME AS PIE)
BE0D 26 05                BNE   CONTIN    IF NOT TOGGLE CHARACTER
BE0F 73 BE24              COM   CAPS      TOGGLE THE FLAG
BE12 4F                   CLRA            DON'T PASS CONTROL CHARACTER TO EDITOR
BE13 39                   RTS
BE14 7D BE24    CONTIN    TST   CAPS      IS FLAG ON?
BE17 27 0A                BEQ   CONT2     DONE IF IT IS OFF
BE19 81 61                CMPA  #'a       ELSE SEE IF CHARACTER IS LOWER CASE
BE1B 25 06                BLO   CONT2     DONE IF TOO SMALL
BE1D 81 7A                CMPA  #'z
BE1F 22 02                BHI   CONT2     DONE IF TOO LARGE
BE21 84 DF                ANDA  #$DF      ELSE MAKE IT UPPER CASE
BE23 39         CONT2     RTS
BE24 00         CAPS      FCB   0
BE25 BE27       INCHV     FDB   INCH
BE27 8D D7      INCH      BSR   INCHNE
BE29 34 02                PSHS  A
                    *
                    * ADDRESS $D3F9 SHOULD CONTAIN THE ADDRESS OF A SUBROUTINE
                    * THAT DOES A JMP (#F00A)
                    *
BE2B AD 9F D3F9           JSR   [$D3F9]
BE2F 35 02                PULS  A
BE31 39                   RTS
                          END

0 ERROR(S) DETECTED
```

OS9 USER NOTES

By: Peter Dibble
517 Goler House
Rochester, NY 14620

As I was working away, distracted by the problem of choosing a topic for this month's column, I deleted a bunch of files by mistake. Worse, I didn't notice that I had done myself in until minutes later -- too late to get the files back. This event made the choice of a subject for this month substantially easier. The first topic for this month is file security.

Users on OS-9 are known by a number. If you use OS-9 as it came off the distribution disk you

will be the only user and have the user number 0. User 0 is special: UNIX users would call him the superuser. The superuser has special privileges that enable him to circumvent the protection of files. All other users, and, to some extent the superuser, are separated from disk files by OS-9's file protection scheme.

If you use the DIR command with the "E" option: OS9: DIR E you will get a list of the files in your current working directory with a lot of information about each file:

```
   directory of .  19:50:32
Owner Last modified  attributes  sector  bytecount  name
----- -------------  ----------  ------  ---------  --------
    1  83/05/10  2234  ------wr      C0       4141  Column4
    1  83/09/11  2351  -s--r-wr      AF       45BD  Column8
    1  83/06/11  1630  ------wr      B4       6081  Column5
    0  83/09/14  2036  d-ewrewr      BD       3E0   PROGRAMS
    1  83/08/28  1614  ----r-wr     245       8E9   Dictionry
```

The information in this display that relates to a file's protection are its owner and attributes. All the files in this directory, except the file (a directory file) called PROGRAMS, belong to user number 1. The type of protection given to a file depends on the contents of the attributes field.

The first position in the attributes field is for the directory attribute. Directory files have several special characteristics, the one relating to protection is that they can't be deleted with the DEL command.

The second position in the attribute field is the sharable attribute. If there is an "s" in this position, the file can only be accessed by one process at a time.

The next six positions in the attribute field are two groups of three attributes each: public execute, write and read, and private execute write and read. If a public attribute is on (indicated by a letter instead of dash in that position) then any user can do that class of operations. If a private attribute is on, the owner of the file can do that class of operation.

The file called Column4 has typical protection. User 1, who is the owner of the file, can read or write it, and nobody else can do anything to it except observe that it is there.

Column8 is protected such that any user can read the file, but only user 1 can write to it. It also has the non-sharable attribute which protects it against being accessed by more than one user at a time. The non-sharable attribute prevents things from getting confusing when user 1 is updating the file and some other user is reading it, by preventing that situation from coming about. Whoever gets to the file first has exclusive access to it until he closes it. If several users want to read a file at the same time there is usually no reason not to let them do so, problems start to appear when a user wants to write to the file while other access it, and things get really sticky if several users want to update the file at the same time. The non-sharable attribute is most important when several users want write to a file concurrently.

The protection of column5 demonstrates one of the more useful applications of file protection in a single user system. It is impossible for anyone, even the owner of the file, to write to it without first changing its attributes. Since the class of operations controlled by write protection includes writing, renaming, and deleting, a file which is write protected can't be deleted by mistake. If I had write protected my files I wouldn't have been able to delete them without thinking about it.

It would appear to be impossible to ever delete a write protected file, but the owner of the file can use the attr command to change the attributes. The procedure for deleting a write protected file is: use the attr command to remove the write protection:
OS9: attr column5 w Then delete the file with the normal del command.

None of the data files in this directory have the execute attribute. They are all text files and manifestly not executable. OS-9 will only load a file for execution if it has the executable attribute. The separation of the execute attribute from the read attribute makes it possible to create an execute-only file. It would be difficult for someone to copy, dump, or disassemble an execute-only file. The execute-only attribute is a useful trick for protecting proprietary software.

A particularly sneaky problem is related to the execute attribute. Merging executable files together to form a file with all the modules used by some program, or to allow a set of popular utilities to be loaded compactly under Level Two, will create a file which doesn't have the execute attribute. OS-9 won't let you execute or load the resulting file. It gives an error 214, "NO PERMISSION." The fix for this problem is to use the ATTR command to give the file the executable attribute.

If you don't intend to have more than one user on your computer there is no reason for you to worry about user numbers. If you want to share your computer with other people -- either taking turns using the computer or using OS-9 as a multi-user operating system -- it is a good idea to have a separate user number for each person who uses the computer. The best way to set your user number is to start the TSMON process in the startup file. The last line in the startup file should be something like:
TSMON /TERM TSMON will just sit there until you type a carriage return. This may give you the impression that something is wrong with the computer unless you are ready for this stolid lack of activity. To comfort myself I include the line:

```
ECHO Press carriage return to initiate logon>/TERM
```

before the TSMON command in the startup file. It leaves directions on the screen after I boot the system. If you are lucky enough to own a system large enough to support three terminals, the following sequence of commands could be included in the startup file to get everything going:

```
Echo Press carriage return to initiate logon >/TERM
Echo Press carriage return to initiate logon >/T1
Echo Press carriage return to initiate logon >/T2
TSMON /T1&
TSMON /T2&
TSMON /TERM
```

It is important to start the last TSMON as a foreground task (no &).
The main business of TSMON is done by the LOGIN command. The LOGIN command uses files called password and motd which must be in the SYS directory

on the same disk the default data directory is on (normally /00). The password file includes the user-name, user-number, and, optionally, a password for each user authorized to use the computer. It also includes a lot of information used to set up the environment for each user. The full contents of each line in the password file are:

User Name
Password
User Number
Initial priority
Initial execution directory (usually .)
Initial data directory (usually .)
Initial program to execute (usually "shell")

The login command prompts for a user-name, and, if that user has a password in the password file, for a password. If the user-name isn't in the password file, or the password isn't correct, LOGIN announces the mistake and prompts for user name again.

The login command protects each user number from unauthorized use by insisting on getting a good user-name/password match before letting someone use a user-number. Many different users can share a user- number, allowing them to share files in a group, but each user-name can only be associated with one user number.

If you find a need to change your user number in the middle of a session with your computer you may be able do it with the LOGIN command. The LOGIN command can only be used if your default data directory is on the same disk the password file is on. The LOGIN command needs to read the password file. If you protect the password file against public read to keep everyone from browsing through the passwords, nobody but the superuser can use the LOGIN command.

The motd file contains the "message of the day." If there is any text in motd it will be displayed on the screen each time anyone logs on. It can be used to display a general greeting, or to give system status information of general interest; e.g., "We are running a new release of Pascal today."

Some tricks can be done with the "initial program" in the password file. It is possible to specify not only the initial program, but also a parameter string for it. This opens up extensive possibilities. Most operating systems allow a user to have the commands in a file (sometimes called a user profile or a login command file) executed every time he logs on. If you are willing to accept some limitations, the initial command can be used to do much more than start a shell for you when you log on.

The simplest possible entry in the password file might go something like:

myname,,3,100,.,.,shell which would set up a user called myname. Myname would have a usernumber of 3, and would be started with a priority of 100. His data and execution directories would be standard -- for most systems /D0 and /D0/CMDS. Whenever myname logs in a shell will be stared for him.

A somewhat more demanding user can make the password file do much more for him. The following line in the password file sets up a user with a password of xyzzy, gives him non-standard data and execution directories, and runs FREE and MFREE for him before leaving him running a shell:

hisname,xyzzy,2,150,/D0/HISDIR,/00/BASICX,
shell free;mfree;ex shell

I had to spilt the line after the execution directory in order to fit it into a column. In the password file it must all be on one line. The important thing is that the sequence of commands the user wants executed must start with the name of the program that will interpret the rest of the line. If that program is the shell, the last command in that shell's parameter string must be an ex for whatever command you want to start the user with.

If you want to start a user with a particularly long script of commands, perhaps enough commands to hold him for an entire session, use a shell command file. The trick is to have the initial command be "shell" with a file name as the parameter. If the file isn't in the default data directory its full path-name must be specified. A sample password file entry might go like:

hername,witrs,5,130,...,shell her.cmd.file ; ex shell

In this case the file "her.cmd.file" must be in the system default data directory. The command file invoked at login is just like any other shell command file. The important restriction to remember is that the shell command file is run by a different shell from the one that the user will be using when the command file is finished. If you change the directories in the command file, those changes will effect only the shell running the commands, not the shell that will be running after the command file is done.

The "Suspend State"

Microware has added a nifty performance enhancement to the latest version of OS-9. They discovered that device drivers were spending a significant amount of time using the F$SEND service request (SR) to communicate between the interrupt service routine for the port and the rest of the device driver. In order to understand why the send was done you need some background in the way the OS-9 SCF device drivers work. The simplest way to write a device driver is to read and write to the port directly from the read and write entries of the driver, but this requires that the driver go into a wait loop while the interface chip is performing the operation. A wait loop isn't a bad thing if the processor has nothing to do until the I/O is complete, but, in an environment like OS-9, there are likely to be several tasks waiting to get done. The "right" way to write a device driver under OS-9 is to have the actual I/O done by an interrupt service routine, and have the read and write entries of the device driver share queues with the interrupt routine.

A character to be written goes to the write entry of the device driver which puts the character into the write queue if there is space for it, or goes to sleep if there isn't. The interface chip should be set to generate an interrupt whenever it is ready to write another character. The interrupt service routine will be started every time an interruot is received from the port it is responsible for. If the interrupt was an output interrupt, the interrupt service routine will take a character out of the output queue and send it to the port. If the device driver is sleeping, waiting for an empty slot in the queue to appear, the interrupt service routine should send it a wakeup signal.

The procedure for reading a character is roughly the reverse of that for writing. The queue for input goes from the interrupt routine to the read entry and the device driver sleeps if a read is done when the queue is empty.

All this sending from the interrupt service routine to the driver is expensive. A new system state called the "Suspend State" was invented to keep device drivers from having to use F$SEND requests to start and stop its read and write

operations. The "suspend state" is a lot like a light nap. The process is in the grey area between sleep and activity. Suspended processes remain in the active process queue where they quickly age to the top of the queue, but while the suspend bit is on in their process descriptor they can't be scheduled. To wake a suspended process up just turn the suspend bit off in its process descriptor. The following code would wake a suspended process from the interrupt routine of a driver:

```
ldx (Address of process descriptor
     for the process you want to awaken)
lda 255-Suspend
anda P$State,X
sta P$State,X This sequence of instructions can
```

be done a great deal faster than a F$SEND.

A process can suspend itself by turning the suspend bit in its process descriptor on, then sleeping for a tick. The sleep is just a way of giving up the rest of the time slice. Even without the F$Sleep, next time the dispatcher sees the suspended process descriptor it will treat it as suspended, and won't start it again until the suspend bit is turned off.

There are a few important limitations to the suspend state. The first is that a process can't get out of suspend state on its own. The second limitation is that the suspend bit is in the process descriptor which is the in the system address space. A non-system process has no easy way of directly modifying the process descriptor. The last limitation is implicit in the advantage of suspend state, suspended processes stay in the active process queue. They will slow the dispatcher down slightly because it will have to pass over them each time it looks for the next process to run.

OS-9 CIS COBOL

Review of OS-9 CIS COBOL

Overview

COBOL is a big language, an old language, and an extremely popular language. Some languages were designed to be compiled and run on small computers; COBOL was not. COBOL is vehemently detested by many people involved with computers, but, despite all the nasty publicity it gets, COBOL is probably the most used computer language in the world. If you need to hire an experienced programmer for a business application, you will find the hunting best if you shoot for a COBOL programmer. COBOL was one of the first compiled languages developed for computers (around 1960), and it has been being (arguably) improved since then. The fully "improved" version of COBOL is an enormous language whose compiler is fully capable of needing the best part of a megabyte of memory to run properly.

There are standards against which any version of COBOL should be measured. ANSI (American National Standards Institute) has defined a COBOL standard which constitutes the official definition of the language. CIS COBOL was written to conform to the ANSI standard definition of COBOL.

To quote the manual: "CIS COBOL is ANSI COBOL as given in 'American National Standard Programming Language COBOL' (ANSI X3.23 1974)." It includes level 1 of the ANSI definition of COBOL along with a few parts of level 2. This doesn't mean that CIS COBOL is the version of the language you may have used on a mainframe computer, but it does mean that if you don't use the enhancements that CIS COBOL includes, the programs you write using it will run essentially unmodified on any other computer that runs level 2 or higher of ANSI COBOL. Also, since CIS COBOL is compiled to intermediate code, programs written in it can be run on any computer that has the appropriate interpreter. If you read the adds in BYTE, you will see that CIS COBOL is implemented for many computers.

I didn't test CIS COBOL exhaustively for conformance to the standard, but I did write a few programs in it. I am used to IBM's VS-COBOL, and a version of UNIVAC COBOL; both are highly enhanced versions of higher levels of ANSI COBOL than CIS COBOL. It took me a while to learn which of my favorite programming tricks aren't possible under level 1 of ANSI COBOL, but, after I learned the limitations I had to live with, I found that I could write programs with no more difficulty than I usually experience when writing in COBOL. I wish I had been able to transfer a program from the IBM to my micro and compile it, but I don't know of any real programs written to be compiled by ANSI level 1 COBOL. Transferring a program in the other direction is no problem.

There is far too much to CIS COBOL for me to say with certainty that it all works, but I understand that the language has actually been successfully tested against a set of standard test programs.

Enhancements

Standard COBOL doesn't support the interactive microcomputer environment very well, but CIS COBOL includes enhancements to the ACCEPT and DISPLAY statements that make it relatively easy to display screens of data, and accept data from fields defined on the screen. Information can be accepted from, or displayed at, a particular cursor location. An input field can be defined as numeric only, in which case any inappropriate characters (like "A") won't be accepted. When a field is filled with data, the cursor automatically jumps to the beginning of the next field. There are special keys which jump the cursor forward and backward a field at a time. Special function keys can be defined. They act like a carriage return (terminate entry into a screen), but a program can determine whether a screen was terminated by a carriage return or a function key, and which function key was used. The location of the cursor when carriage return was pressed is also available. The net effect of these enhancements is that it is fairly easy to write CIS COBOL programs that accept and display screens of data.

In addition to the usual COBOL file organizations (including ISAM), CIS COBOL allows an organization they call "line sequential." Line sequential files are variable length record files, in which the records are terminated by carriage returns. This makes it easy to read and write files that Pascal would call "files of text." The most generally important examples of files of this type are files created by text editors, and line by line output to a terminal or printer. The other access modes supported by CIS COBOL are: sequential, relative, and indexed.

The names of files can be specified at run time using statements like:

```
SELECT FILE-15 ASSIGN TO FILE-15-NAME.
...
ACCEPT FILE-15-NAME.
OPEN INPUT FILE-15.
```

In addition to the standard ANSI debug features, CIS COBOL has a respectable interactive debugger. The commands available under this debugger are:

P - Display the current program counter
G - Set a breakpoint
X - Single step
D - Display data at specified offset in data division
A - Change memory (ASCII)
S - Set block for display or change
/ - Display block
. - Change bytes in block
T - Trace paragraphs
L - Write CR,LF
M - Define a debug macro
$ - End a macro definition
C - Display a specified character
; - precedes a comment (for describing macros)

The interactive debugger can be used on any COBOL program by including +D on the command line that invokes the program, e.g., RunC +D test.int. This means that you can use the debugger on a program without having to do anything special when you compile it.

Microware has included eight subroutines in the COBOL run time system which can be called from a COBOL program. MOVE-BLOCK is a procedure that can be used to do a high speed move of a block of data. ABORT terminates the program with an error code. CHAIN makes the standard OS-9 F$Chain system call available. The FUN-KEY subroutine can be used after a ACCEPT statement to find out if a function key was pressed instead of the carriage return key, and which one. DATE returns the date and, optionally, the time. SHELL invokes a shell, passing it a specified string. CHX and CHD change the execution and data directories for the program.

The subroutines in the run time system are called by number. CIS COBOL can also call subroutines which are either COBOL I-code, or object code. The CALL statement looks like:

CALL "/DO/SUBLIB/TEST.SUB.1" USING ...
ON OVERFLOW .... The called program is loaded into memory if it is not already there, and, depending on whether the module header indicates that it is I-code or object code, interpreted or executed. If there is no room in memory for the new module, the ON OVERFLOW clause in the CALL statement gets control. The CANCEL verb unlinks a subroutine, freeing the memory it is using.

In addition to these methods of calling external subroutines, CIS COBOL supports program segmentation, which can be used to divide the program into sections that will remain on disk until they are needed. Segments use memory efficiently at the cost of extra disk I/O by sharing a common pool of overlay memory.

In addition to supporting ANSI COBOL level 1, including:

The Nucleus
Table Handling
Sequential Input and Output
Relative Input and Output
Indexed Input and Output
Segmentation
Library (Copy)
Inter-program communication

debug CIS COBOL supports parts of level 2 of ANSI COBOL including:

Nested IF
PERFORM UNTIL
The START statement for Relative and Indexed I/O and full level 2 inter-program communication

Limitations

I was disappointed with some of the restrictions of the low level of COBOL implemented for CIS COBOL, but not very surprised. I am more upset by some problems with terminal support, and the CONFIG utility that is used to customize the run time package for a particular type of terminal.

The features of advanced levels of COBOL that I missed most were AND and OR in IF statements. It is possible to do without boolean operations in IF statements, but I am not used to having to work around a limitation like that. Another very popular feature which is missing in CIS COBOL is the SORT statement. A surprising number of production COBOL programs include at least one sort, and it would be hard to eliminate a sort from a program without a major redesign.

The run time system which interprets the COBOL intermediate code also includes routines for terminal control. It is customized for a terminal by a utility program called CONFIG. I was not impressed with CONFIG. My favorite terminal uses the ANSI standard terminal control sequences ... CONFIG was clearly not written with my terminal in mind. I struggled for two evenings trying to get RunC configured for my TeleVideo with no success. Finally, I gave up and turned to my H-19, which was much more like what CONFIG wanted ... I had COBOL running in ten minutes. There were three fundamental problems with CONFIG's handling of my TeleVideo's control sequences. CONFIG expected most terminal control strings to be no more than three characters long; several of the ANSI strings are longer than that. CONFIG simply can't deal with the ANSI direct cursor positioning sequence; I circumvented that problem by pretending that my terminal didn't have a direct cursor positioning command, and specifying relative positioning. CONFIG can only deal with commands that move the cursor one row or column at a time in relative positioning mode. Since the ANSI strings that cause the cursor to move one row or column are three characters long, this is a slow way to adjust the cursor position. The clear-screen sequence for my terminal is four characters long; so I couldn't use it. RunC tries to fake a clear-screen somehow, but it makes a real mess of it. The clear-screen sequence somehow came out as a string of thousands of <bell> characters. I understand that a more recent version of CONFIG than the one I have allows a four character string for the clear-screen sequence. I think that would have made it possible for me to get my TeleVideo working with COBOL.

CONFIG forms a trap for the unwary user. Once you start into it there is no turning back. If you change your mind about the response you just keyed in, you have to wait until you reach the end of the entire (long) string of questions, and ask to be allowed to change a large subset of your answers. When you are going through CONFIG to fix a mistake or change an existing terminal description to fit a new terminal, you have to fill in the correct answer to each question. There is no way to select a default, or keep the old value. It is true that CONFIG is not likely to be a heavily used utility, but I found it so hard to use that I would much rather have written a few subroutines to support my terminals.

Once I got the screen support working, I found that I wasn't pleased with the way it worked. I believe that when the cursor leaves a numeric field, the field should be right justified and zero filled. The screen handling package in CIS COBOL seems to agree with me to some extent. If you enter a "." in an integer field it will right justify and zero fill, but if you exit the field with a carriage return (ending the entire screen) or down arrow

(moving to the next field), a test for numeric in the program will indicate that the field is not numeric. If the field has editing characters in it the field is inclined to end up left justified and zero filled.

I am used to getting useful, english error messages from COBOL; CIS COBOL gives error messages with numeric codes in them indicating what the error is. Even after I looked up the error, it wasn't clear what the problem was. For instance, when I hadn't declared a variable it told me that there was a type mismatch in the statement using the undeclared variable. When I tried to use AND and OR, it gave me the same error. I ended up treating the error message as "something's wrong around here."

Benchmarks

I ran two benchmarks against this COBOL: one for speed at numeric processing (the sieve), the other for speed in handling ISAM files. I adjusted the prime number program from the January 1983 BYTE slightly to fit ANSI level one, and ran it. This version of COBOL would have fallen nearly at the bottom of the chart given in that BYTE, between Microsoft COBOL and RMCOBOL. It took 541 seconds to find the first 1899 primes. I could have made the program run somewhat faster by using indexing instead of subscripting, but that would have spoiled the benchmark. I have to admit that I felt silly writing a Eratosthenes Sieve program in COBOL. Testing COBOL for its ability to find prime numbers is like testing programmers for their ability to read Latin; they may be able to do it but it is hardly relevant. I ran that benchmark because it is the most used benchmark for microcomputer languages, but I also ran another non-standard, but, I think, more relevant, benchmark.

I constructed a benchmark program which gives a good measure of the speed with which the language handles indexed I/O. Indexed I/O is very important to the group of users who might use COBOL. Interpreting the results of a benchmark that involves I/O is a little tricky. Certainly the file structure the language uses is very important, especially with a large indexed file; but the access time for the disk is an important factor, and the time the operating system takes for a context switch is somewhat important.

I built a file 10,000 records long of 55 byte records with five byte keys and then read it randomly reading two records alternately from each end. It took 2615 seconds to build the file and 3233 seconds to read the file (it would, of course, have been possible to read it faster if I had read sequentially). I ran these benchmarks on a GIMIX system with a CM 5000 Winchester (a file that size would not have fit on my 8" floppies). I used OS-9 Level Two on a 2 mhz 6809. The performance would have been much worse if I had used a floppy instead of a Winchester, and somewhat better if I had used GMX-III.

I compiled three COBOL programs on the same machine I ran the benchmarks on. A simple merge program which I haven't included with this review took 45 seconds to compile, the sieve compiled in 35 seconds, and the ISAM test program took 43 seconds.

Summary

It is possible to get past the problems with CONFIG, to learn to live with the primitive error messages, and to feel comfortable with the screen handling conventions. What is left is a substantial implementation of an old, but useful language. I don't think everyone should run out and buy this package, but, for a few people, it could be uniquely useful. If you want to use a group of COBOL programs on microcomputer, it would certainly be easier to convert them from one level of COBOL to another than to translate them into an entirely different language. CIS COBOL would be a good teaching tool for schools unable to afford time on a machine with a full-blown COBOL compiler. It should be relatively easy to find programmers who can work in COBOL. With CIS COBOL, a microcomputer could be used as a development environment for COBOL programs, though the low level of CIS COBOL would prevent this in most cases. Perhaps the most significant advantage of CIS COBOL over other languages is that programs written in CIS COBOL can be moved in I-Code form to a variety of other machines and operating systems, and run without source code. UCSD Pascal has shown that this is an asset even though it can't generally run under a normal operating system.

CIS COBOL was written by Micro Focus Limited. Microware wrote a run time package for it that allows any program written in CIS COBOL, including CIS COBOL itself, to be run under OS-9. By writing a run time package for CIS COBOL, and arranging to license it for OS-9, Microware made a large collection of business software available to OS-9 users. If you are looking for a nice accounting system, payroll, MRP system, or whatever, check with Microware. They have a long list of vendors offering programs which run under the CIS COBOL run time system.

Some small number of people will find Microware's version of CIS COBOL just what they need. If you think you are one of those people, I recommend that you get the manual before you commit to the language. The manuals won't be any help to you if you don't know COBOL, but, if you do, they will leave you with an accurate impression of the language, and either leave you impatient to get the software, or disappointed about some important missing feature (most likely sort).

Peter Dibble

\- - -

```
** CIS COBOL V4.4                    siev.cbl                    PAGE: 0001
**
       IDENTIFICATION DIVISION.                                        0110
       PROGRAM-ID. SIEVE.                                              0110
       AUTHOR. PETER DIBBLE.                                           0110
       ENVIRONMENT DIVISION.                                           0110
       WORKING-STORAGE SECTION.                                        0110
       77  PRIME                       PIC 9(5) COMP.                 0184 00
       77  PRIME-COUNT                 PIC 9(5) COMP.                 0187 03
       77  I                           PIC 9(4) COMP.                 018A 06
       77  K                           PIC 9(5) COMP.                 018C 08
       01  BIT-ARRAY.                                                 018F 00
           03  BIT OCCURS 8191 TIMES   PIC 9 COMP.                    018F 08
       PROCEDURE DIVISION.                                             0000
       START-UP.                                                       0000 00
           DISPLAY 'TEN ITERATIONS'.                                   0009
           PERFORM SIEVE THROUGH SIEVE-END.                            0020
           DISPLAY 'PRIMES FOUND: ', PRIME-COUNT.                      0025
           STOP RUN.                                                   0044
       SIEVE.                                                          0045 00
           MOVE ZERO TO PRIME-COUNT.                                   0046
           MOVE 1 TO I.                                                0048
           PERFORM INIT-BITS 8191 TIMES.                               0055
           MOVE 1 TO I.                                                0061
           PERFORM SCAN-FOR-PRIMES THROUGH END-SCAN-FOR-PRIMES         0069
               8191 TIMES.                                             0069
       SIEVE-END.                                                      0077 00
           EXIT                                                        0078
       INIT-BITS.                                                      0078 00
           MOVE 1 TO BIT (I).                                          0079
```

```
    ADD I TO I.                                          0089
END-INIT-BITS.                                           0097 00
    EXIT.                                                0098
SCAN-FOR-PRIMES.                                         0098 00
    IF BIT (I) = 0                                       0099
        THEN                                             00AC
            GO TO NOT-PRIME.                             00AC
    ADD I I GIVING PRIME.                                00B0
*   DISPLAY PRIME.
    ADD I PRIME GIVING K.                                00C3
    PERFORM STRIKOUT UNTIL K > 8191.                     0001
    ADD 1 TO PRIME-COUNT.                                00E6
NOT-PRIME.                                               00F4 00
    ADD 1 TO I.                                          00F5
END-SCAN-FOR-PRIMES.                                     0103 00
    EXIT.                                                0104
STRIKOUT.                                                0104 00
    MOVE 0 TO BIT (K).                                   0105
    ADD PRIME TO K.                                      0115
END-PROGRAM.                                             0123 00
    EXIT.                                                0124

** CIS COBOL V4.4 REVISION 0                                    URN rp/1841/A
** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
** ERRORS=00000 DATA=08590 CODE=00370 DICT=00272:01611/01883 GSA FLAGS= OFF

** CIS COBOL V4.4                    Bnch.CBL                        PAGE: 0001
**
       IDENTIFICATION DIVISION.                                      0110
       PROGRAM-ID. ISAM-BENCHMARK                                    0110
       AUTHOR. PETER DIBBLE.                                         0110
       ENVIRONMENT DIVISION.                                         0110
       CONFIGURATION SECTION.                                        0110
       SOURCE-COMPUTER. GIMIX.                                       0110
       OBJECT-COMPUTER. GIMIX.                                       0110
       INPUT-OUTPUT SECTION.                                         0110
       FILE-CONTROL.                                                 0110
           SELECT ISAM-FILE-1 ASSIGN "ISAM.FILE";                    0184
               ORGANIZATION IS INDEXED;                              0186
               ACCESS MODE IS SEQUENTIAL;                            0186
               RECORD KEY IS ISAM-KEY-1.                             0186
           SELECT ISAM-FILE-2 ASSIGN "ISAM.FILE";                    01BF
               ORGANIZATION IS INDEXED;                              01C1
               ACCESS MODE IS RANDOM;                                01CI
               RECORD KEY IS ISAM-KEY-2.                             01CI
       DATA DIVISION.                                                01FA
       FILE SECTION.                                                 01FA
       FD  ISAM-FILE-1;                                              01FA
           DATA RECORD ISAM-RECORD-1.                                01FA
       01  ISAM-RECORD-1.                                            01FA
           03  ISAM-KEY-1              PIC 9(9) COMP-3.              01FA
           03  FILLER                  PIC X(50).                    01FF
       FD  ISAM-FILE-2;                                              0253
           DATA RECORD ISAM-RECORD-2.                                0233
       01  ISAM-RECORD-2.                                            0233
           03  ISAM-KEY-2              PIC 9(9) COMP-3.              0233
           03  FILLER                  PIC X(50).                    0238
       WORKING-STORAGE SECTION.                                      026C
       77  KEY-NO                      PIC 9(9) COMP-3 VALUE 0.      026C 00
       77  HI-NUMBER                   PIC 9(9) COMP-3.              0271 05
       77  LO-NUMBER                   PIC 9(9) COMP-3.              0276 0A
       77  DATE                        PIC XXX VALUE "004".          027D 0F
       01  WORK-DATA.                                                027E 12
           03  WORK-KEY                PIC 9(9) COMP-3.              027E 12
           03  I-DATA                  PIC X(50).                    0283 17
       01  SYSTEM-DATE.                                              02B5 49
           03  YEAR                    PIC 99.                       02B5 49
           03  MONTH                   PIC 99.                       02B7 4B
           03  DAY                     PIC 99.                       02B9 4D
       01  SYSTEM-TIME.                                              02BB 4F
           03  HOUR                    PIC 99.                       02BB 4F
           03  MINUTE                  PIC 99.                       02BD 51
           03  SECOND                  PIC 99.                       02BF 53

       PROCEDURE DIVISION.                                           0000
       START-UP.                                                     002C 00
           OPEN OUTPUT ISAM-FILE-1.                                  002D
           MOVE "ASSORTED DATA: NAME, ADDRESS, ETC, OR WHATEVER" TO  0031
               I-DATA.                                               0031
           ADD 1 KEY-NO GIVING LO-NUMBER.                            0065
           MOVE KEY-NO TO WORK-KEY.                                  0073
           DISPLAY "START BUILD".                                    007A
           CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.                 008E
           DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND            0096
           PERFORM ADD-RECORD 10000 TIMES.                           009F
           CLOSE ISAM-FILE-1                                         00B7
           DISPLAY "BUILD DONE".                                     0007
** CIS COBOL V4.4                    Bnch.CBL                        PAGE: 0002
**
           CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.                 00EE
           DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND            00FA
           MOVE WORK-KEY TO HI-NUMBER.                               011F
           DISPLAY "READ STARTING".                                  0131
           OPEN INPUT ISAM-FILE-2.                                   0147
           PERFORM TEST-READS 2500 TIMES.                            0148
           CLOSE ISAM-FILE-2.                                        0157
           CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.                 015B
           DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND            0163
           DISPLAY "READ DONE".                                      018C
           STOP RUN.                                                 01A9
       ADD-RECORD.                                                   01AA 00
           ADD 1 TO WORK-KEY.                                        01AB
           WRITE ISAM-RECORD-1 FROM WORK-DATA;                       01B9
               INVALID KEY PERFORM ERROR-1.                          01C1
       ERROR-1.                                                      01CD 00
           DISPLAY "INVALID KEY: ", ISAM-KEY-1.                      01CE
       TEST-READS.                                                   01ED 00
           PERFORM READ-HIGH.                                        01EE
           PERFORM READ-HIGH.                                        01F1
           PERFORM READ-LOW.                                         01FA
           PERFORM READ-LOW.                                         01F7
       READ-HIGH.                                                    01FA 00
           MOVE HI-NUMBER TO ISAM-KEY-2, WORK-KEY.                   01FB
           READ ISAM-FILE-2; INVALID KEY PERFORM ERROR-2.            0208
           SUBTRACT 1 FROM WORK-KEY GIVING HI-NUMBER.                0217
       ERROR-2.                                                      0225 00
           DISPLAY "INVALID KEY: ", WORK-KEY.                        0226
       READ-LOW.                                                     0244 00
           MOVE LO-NUMBER TO WORK-KEY, ISAM-KEY-2.                   0245
           READ ISAM-FILE-2; INVALID KEY PERFORM ERROR-2.            0255
           ADD 1 WORK-KEY GIVING LO-NUMBER.                          0261
       END-PROGRAM.                                                  026F 00
           EXIT.                                                     0270
** CIS COBOL V4.4 REVISION 0                                    URN rp/1841/A
** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
** ERRORS=00000 DATA=00705 CODE=00703 DICT=00612:01271/01883 GSA FLAGS= OFF
```

"C" User Notes

Norm Cramer
3 Pryor Road
Natick, MA 01760

As promised last month, we will have look at the Dyna-C compiler; also, some hints on using command line arguments and, since I have gotten some letters about the subject, some of the different ways you can play around with arrays and pointers.

DYNA-C COMPILER

I finally got some time to investigate the OS-9 version of the Dyna-C compiler and, so far, have liked what I've found. The compiler is a Small-C variant for the 6809 based on Ron Cain's original article; which is freely acknowledged in the introduction of the manual.

The whole file package fits onto a single 5.25" diskette and includes

- manual
- Dc — the compiler (DC.CMD for FLEX)
- chead.a — a standard runtime support header
- stdio.h — the standard I/O header file
- stdio.a — the assembler I/O file
- extio.c — extended I/O functions and string operators
- ctype.h — an assembler file containing such functions as toupper(), tolower() and such
- primes.c

The manual is quite well done. The organization and neat layout make for easy, comprehensive *reading*. After a brief introduction and description of the distribution disk, it launches right into how to compile programs. Next comes the Dyna-C language definition. This section details the differences between Dyna-C and the full implementation. There is a section on the standard library, running separate compilations and compiler internals. The last section is summary of Dyna-C syntax.

I've said it so often that this will probably sound like a broken record. The manual does not attempt to give a tutorial on the language. But it does provide adequate documentation to understand and use the compiler.

The compiler is a pretty good subset for a "small" compiler. It would probably be shorter to say what it doesn't have, but safer to tell you what it does have. Here's a partial description of its features

data types
    int, char

declarations
    type - int or char,
    pointer to type - *type,
    array of type - type[],
    array of pointer to type - *type[],
    pointer to pointer - **type,
    pointer to array fo pointers - **type[]
    etc...

constants
    decimal, octal, hex, character, string

escaped characters
    The usual complement with one exception; \n
    the newline character is really <cr>. If
    you want <lf> then you must use \010.

operators
    The operators include just about the full
    span of unary, binary, assingment and
    logical operators. Two that are missing are
    the >>= and <<= assignments. On the other
    hand, the conditional assignment operator
    ?: is included.

control structures
    if-then-else
    while
    do while
    for
    switch
    break
    continue
    goto

directives
    #include
    #define
    #ifdef-#ifndef-#endif
    #asm-#endasm

As you can see, it's a pretty complete set that should allow you to go quite far in general, system type programming before the compiler starts to limit you in any severe way.

Compiling a program is pretty straight forward. If the whole program exists in one file then you can get by with the commands (for OS-9)

```
Dc [options] sourcefile asmfile [sourcefile2]...
Asm asmfile o=objectfile #20k
```

The first line would actually compile the input files and produce an output file. If no assembler file name is given, then the output goes to the terminal, in which case you could only compile one file since the second source file would be considered the assembler file. The next command just assembles the file. The #20k is an OS-9 parameter that tells the operating system how much data memory to allocate to the process.

There isn't anything magic about the 20k. The number was taken right from the example in the manual. I tried one compilation of a reasonably sized program (about 6 or 7 pages of C source code) without specifying any additional memory and the assembler ran out of symbol table. Using 20k allowed the program to assemble without a hitch.

There are only two or three switches (options) for the compiler, depending on the operating system. These are

-s Include the C source code as assembler comments.

-pxx Indicates that this module is not the "main" module so that internal compiler labels will start with the prefix xx.

-nName (OS-9 only) assign Name as the module name. The default is Cprog, or any name specified in the #module directive.

The -pxx is a really nice way to handle multiple modules when you can't use a linker. Most Small-C compilers allow you to specify the starting number nnnn for each module of the program. This could get touchy when there are a number of modules. You have to start with incr ents big enough to avoid duplicate labels. It is also a VERY sticky problem when you to build up decent libraries.

Dyna-C clears away most of the problem by letting you pick the prefix instead. Even avoiding the use of "cc", which is the default, still leaves LOTS of different combinations. In fact, it may often be possible to pick to letters that are mnemonically meaningful to the file. This could make generating and maintaining libraries easier. Kudos to DynaSoft for a clever idea on that one.

Where I personally differ with DynaSoft is how they include standard support functions. To me the organization of the standard library seems chaotic. For example, fopen(), fclose(), read() and write() are in chead.a. Getchar(), getc(), putchar(), and putc() are in stdio.a; which gets called from stdio.h. The file extio.c has the functions printf(), fputs(), gets(), fgets(), fputs(), strcat(), strcpy(), strlen(), and atoi(). Finally, ctype.h includes most of the is...() and to...() functions which are written in assembler between #asm and #endasm directives.

A more convenient approach might be to make the entire library "stand alone" assembler sources with a finer granulation of functions within files, according to their use. A potential reorganization might be

chead.a All necessary run time set up and the terminal primitives getchar(), gets(), putchar() and puts().

extio.a File I/O such as fopen(), fclose(), getc(), fgets(), putc(), read(), write(), and fputs().

char.a The character test functions is...() and to...().

string.a The basic string functions such as strcat(), atoi(), strlen() etc.

printf.a The formatted I/O routines such as printf(), fprintf() etc.

I realize that there may be coupling between many of the functions that would render such an arbitrary

grouping impractical. But there are advantages to doing it this way. The two most important would be that the inclusion of the library can be made independent of any C source modules and that it will occur at assembly time, not at compile time. This could result in significant time and code size savings. What would then be required is an assembler file that contains nothing more that a bunch of INCLUDE or LIB statement that pull in all the necessary modules and libraries. This should sound familiar; I have mentioned it for other Small-C compilers as well.

Note that the sources for much of the library could be supplied to the user as C code, who could then modify them to suit the needs at hand.

This approach is not without its own disadvantages of course. Careful attention must be paid to the assembler file that contains all the INCLUDE's to insure that all the necessary modules are brought in. Adding a new module would require updating that file. It might also require a more thought when transporting programs from other environments , such as UNIX.

For example. Under UNIX, the is...() and to...() character test functions are really macro's contained in the file ctype.h (I rather suspect that this is the reason Dyna-C has them as assembler code the the same file). If you had implemented a library along my suggestions, you would certainly have to do some editing in order to compile the program. Oh well, there goes the free lunch!

Getting back on the track. When I received the package I also got some of Dynasoft's proprietary utilities to try on the compiler. I have compiled and run these programs as well as the old standby primes.c.

The compilation and assembler process by and large went smoothly and without a hitch. I did have to slightly modify one C program by changing the order of two #includes to avoid one error, but that was all.

The programs seemed to work smoothly, which I expected since they are in use at Dynasoft all the time. Examination of the code produced by the compiler showed it to be tight and well structured. Primes ran in 29.5 seconds on my 1Mhz system. That's pretty quick. In fact, I believe that it's the fastest time I've seen from a Small-C compiler for this coding of the primes program. And that includes the 4Mhz Z80's as well. Come to think of it, at 2Mhz (equivalent to a 4Mhz Z80) the program should run in a little under 15 seconds. That even beats most of the big compilers for the Z80 according to the benchmarks published in BYTE magazine last year.

Based on the programs I've compiled and the code I've looked at I would rate this as a very desirable package for the C neophyte. You can get your feet wet without a lot of expense and still do a lot of very serious programming before needing a full compiler.

COMMAND LINE ARGUMENTS, ARRAYS AND POINTERS

I have received a few letters that requested some more details on pointers, arrays and strings. Some of you have also wanted some hints on using command line arguments. Since command line arguments are strings, which in turn are accessed as arrays or with pointers, let's kill two birds with one stone. We'll look at strings, pointers and arrays by using command line arguments.

The processing of command line arguments is really a function of both the operating system and the particular compiler that you are using. All the 6809 compilers on the market today have some sort of argument handling as part of the runtime support code.

The minimum handling consists of converting all the arguments (of the command line that invoked the program) into valid, NULL terminated C arrays of characters, or strings. This can happen in a number of ways. The compiler could first copy the whole line into another buffer or do the conversion with the line still contained in the operating system's command line buffer. Where it actually happens is really irrelevant to the program.

Three things happen as each argument on the line is processed. First, an argument count (initialized at zero) is incremented. Then a pointer to the first character of the argument is put into an array of pointers. Finally, the end of the argument is found and the character following it is changed to a NULL, thereby terminating the string.

After the command line is parsed main() is called with two parameters on the stack, the count of the arguments and the address of the first pointer (of the array of pointers).

Let's assume we are running a program under FLEX where the runtime code parses the line in FLEX's command line buffer, which starts at $C080. Suppose that the command line is

```
+++cprog arg1 arg2 arg3
```

For convenience, let's put a ruler under the line to count off the characters in the buffer. The ruler will be calibrated in Hex to make life easier.

```
cprog arg1 arg2 arg3
-------+-------+-------+
0      8       10
```

By a convention established with the UNIX operating system, the first argument is the program name. So the argument count starts at 1. Let's also assume that the program is building the array of pointers in memory at $4000.

The parser would start scanning the command line from $C080 looking for the first argument. In this case it would encounter 'c' at $C080. It would then increment the arg count to 1; put address $C080 into locations $4000 and $4001; then scan for and change the terminator following CPROG into a NULL. Since the terminator was not a <cr>, it would continue scanning until it came across arg1. It would then repeat all the previous steps. This process would continue until the end of the line terminator <cr> was encountered.

The final argument count would be 4. If we stopped the program at this point and did an EXAMINE of $4000 from SBUG, the first 8 bytes would look like

```
$4000  C0 80 C0 86 C0 8B C0 90
```

These are the pointers to each argument on the line.

The argument count, 4, and a pointer to the first element of the array, $4000, are put on the stack just prior to calling the user's program. Note that it is the address of the pointer array that is put on the stack and not the address of the command line.

To get access to these arguments you must declare main as

```
main(argc, argv)
   int argc;      /* the argument count */
   char *argv[];  /* the pointer array */
```

There is nothing sacred about using the names argc and argv for the variables. This is also a convention carried over from C programs developed on the UNIX system. You can name them anything you want, it has nothing to do with the language.

Understanding the declaration for argv may not be obvious to the casual or beginning user so let's decipher it. This is done by starting at the variable's name and working outward.

> There is a variable named argv. The name is enclosed by * and [].
>
> The [] is a primary expression operator and has the highest binding priority. Another way of saying that you attach it to the name first.
>
> argv[]
> an array (of something)
>
> The * could either be the multiply operator or indirection operator. Since this is a declaration, it must be the latter. We now have
>
> *argv[]
> an array -- of pointers (to something)
>
> Finally, we are left with char. So we end up with
>
> char *argv[];
> an array -- of pointers -- to chars

Another, synonymous, declaration form for argv would be

```
char **argv;
a pointer -- to a pointer -- to chars
```

It may take you a while to accept these as being equivalent, but the compiler treats them that way.

Ok, so how do we use all this stuff in a program? Consider the case where a program expects to find a list of one or more file names to process and possibly an option or two. For purposes of the following discussion assume the previous declaration for main(); repeated here for clarity.

```
main(argc, argv)
   int argc;
   char *argv[];
{
   ...
```

Since the program needs at least one file name to process, calling the program with no arguments is an error. We can handle this condition with

```
if (argc < 2)
    {
    printf("no filenames");
    exit();
    }
```

If argc is 2 or larger, then we have one or more arguments. Now we must check whether any particular argument is an option or a filename. This could be done with a simple for loop.

```
for (i = 1; i < argc; i++)
   if (*argv[i] == '+')
       set_option(argv[i]);
   else process_file(argv[i]);
```

Note two things here. The array is an array of pointers. To look at what an array element points to use the form

```
*argv[i]
```

To pass the pointer to a function we just need to pass the actual value of an array element with

```
argv[i]
```

If there was only one option, named x then we might want to test for it right in the loop, in which case the code could be

```
for (i = 1; i < argc; i++)
   if (*argv[i] == '+')
       if (*++argv[i] == 'x')
           do_something();
       else
           {
           printf("unknown option");
           exit();
           }
   else process_file(argv[i]);
```

This starts to get a little more complex. Once there is an option, we must look at the second character to determine if is x. The easiest way to do this is to preincrement the pointer past the "+" and test the next character. This is done in

```
*++argv[i].
```

We know that [i] binds the tightest, but will we increment argv[i], or what it points to. Since * and ++ have the same binding lever, but associate from right to left, the rightmost symbol binds first. So we are incrementing argv[i] and then using it as a pointer. If you had not been sure then you could have parenthesised it as

```
*(++argv[i])
```

I tend to parenthesis often since it is very clear both to me and to the compiler what I really want. You can really get carried away with this. Suppose that what is pointed to will be incremented and then the pointer will be incremented. This yields

```
++*++argv[i]
```

Then to add to the confusion, you might also want to preincrement the index which gives

```
++*++argv[++i]
```

Timeout! Enough is enough. The eye-brain system starts to overload on this one. It's cute and the compiler should have no problem interpreting it. But would you really want to try to understand a program sprinkled with these kind of bombshells through it? If the statement becomes too complex break it into pieces.

If you had a pointer to an array it would, by definition, point to the zeroeth (first) element. In the C compilers that we have for the 6809, an array is never passed as a whole entity. Instead we either pass the value or the address of a particular element. To pass the value of element 4 of the array 'value' we would use

```
do_something(value[4]);
```

But to pass its address we would use

```
do_something(&value[4]);
```

It just so happens that, as a convenience, we can refer to the address of the first element of an array by the array name itself. Assume a program has a buffer declared as

```
char user_input[50];
```

The buffer is to be passed to some function that will do something to the line. The code could be

```
act_on_buffer(user_input);
```

This would be exactly the same as

```
act_on_line(&user_input[0]);
```

There are some other ways in which pointers and arrays may be intermixed in use. Generally, it is more efficient to use pointers than arrays since there is less address computation.

Consider the following simple function.

```
process(s)
   char s;
   {
   while (*s)
      switch(*s++)
         {
         case 'a' :  action1();
                     break;

         case 'b' :  action2();
                     break;

         case 'c' :  action3();
                     break;

         case 'd' :  if (s[-2] == 'a')
                        action4();
                     else
                        action5();
                     break;
         default :   action6();
         }
   }
```

Process() has only one argument, s, which is a pointer to char. The function scans the character string and does some kind of action depending on what character is found. For efficiency, s is incremented directly each time through the loop after its value has been taken by the switch statement. There is a unique action for all characters except 'd', which was two possible actions. What action gets taken depends on the character that preceded 'd'. Rather than actually manipulate the pointer and then have to restore it, the function just looks backward with the negative index. In this case, since the pointer has been incremented past the 'd', it is necessary to use -2 instead of -1.

This function is a poor example in that it makes the assumption that a 'd' will never occur as the first character of the line, otherwise it would memory that preceded the first element of the string. However, it does at least bring out the fact that we can use a negative index. Note that s was declared as

```
char *s;
```

but we turned around and used

```
s[-2]
```

This is possible because the compiler looks at the declarations

```
char *s   and   char s[];
```

synonymously. Maybe it's clearer to think of the second declaration as being another way of saying "s is a pointer to char".

You should also get comfortable with the fact that, given an array called 'an_array', then accessing the 5th element of the array can be done with either

```
an_array[5];  or  *(an_array + 5);
```

To understand why think of it this way. The intent of either statement is to look at element 5. So in both case, the compiler will multiply 5 by the size of an array's data type, expressed in bytes. If the data type was char, then nothing would be done. If the data type was int, then the offset is multiplied by two.

Being able to refer to an object as an array or with via a pointer is quite handy. But like all good things it can be abused. A computer scientist might find this lack of strong typing detrimental, but a programmer will find it timesaving.

IT'S A WRAP

That's it for this column. I hope that it has help shed some more light on array and pointers. I don't have the next column nailed down, but I will try to finish up a couple utilities and get at least one of them into print. Tell then...

SELECTRIC 6800 DRIVERS

Marc I. Leavey, M.D.
WA3AJR
4006 Winlee Road
Randallstown, Maryland
21133

Add A Selectric Printer To Your 6800

One of the early realizations I had while setting up my computer for business use was that the ASR-33 just would not make it as a printer. I mean, for program listings, OK, but could you see my editor's face if I sent in a manuscript prepared on it? That kind of "tsoris" I don't need. A friend had a dot-matrix printer which I borrowed for a while. It was fast, but hardly the quality I had hoped for.

Thus it was with quite some joy that I acquired a Selectric printer, with "serial" interfacing. Only problem was that try as I might, no print would come out when I hooked it up to an ACIA port on my 6800 based computer. Some research provided the answer. This Selectric, as many others, uses Correspondence code, not ASCII. Not only is the code itself different, but the baud rate is an oddball (for most computers) 134.5 baud. Hoo boy!

I recalled seeing an ad for an ASCII board for the printer on hand, a Trendata 1000 (identical to the Datatrans 1000). Scrounging through some old issues of Kilobaud, I found it. An assembled ASCII to Correspondence code converter, designed for the Trendata, only $250! Two-hundred and fifty dollars? There has got to be a cheaper way.

Well, of course, there is. Why would I write an article if all you had to do was plunk down a quarter-thou to be on ASCII? Besides, the way I ended up, not only can you switch type elements (golf balls), but the printer even has some smarts, with form feeds and variable length pages. Interested? Read on.

You see, old issues of Kilobaud Microcomputing, 73, and who knows, maybe even Byte, do have value. For while I was routing around looking for that old ad I came across an article in the November, 1979, issue of Kilobaud Microcomputing which described an ASCII to Selectric driver for an 8080 based system. While not directly applicable, it started me thinking, and I examined the author's encoding technique carefully. After some tinkering, both with hardware and software, I came upon a workable scheme. A few modifications later, a very nice printer was up and running.

As the author of the 1979 article pointed out, the scheme used by IBM in the six bit Correspondence code is analogous to the common Teletype(R) five-level Baudot code, in that two character sets are used, with up-shift and down-shift commands. In my 73 Magazine column, RTTY Loop, I outlined a scheme for the 6800 which performed the translating of Baudot to ASCII back in July, 1978. Changing the

timing loops and code table quickly produced the skeleton of a working system.

The code table is organized much as the one in the 8080 article. If you have access to a Selectric manual, one was purchased with my printer and is an invaluable asset, you will find that each character requires six data bits, a parity bit, and start and stop bits. The table is organized so that shifting the bits out of the byte retrieved from the right (LSB) side results in transmission in the correct order. This produces a code table in which the data is in reverse order, on a bit-by-bit basis within each byte, from the standard IBM code. Also, the leftmost bit of each byte is used as a case bit, with "1" stored for lower case and "0" for upper case.

Of course, most of the characters easily sort into upper or lower case. There are three, however, in which case does not matter, at least with the "standard" type ball. These are the space, period, and comma, all of which are represented in both upper and lower case. Rather than have the machine jump back and forth to change cases when encountering one of these characters, I included a bit of logic to bypass the shift change if one of these three characters was to be printed.

Another problem was that of carriage returns. Not to decode line feeds was obvious. The Selectric has no mechanism for performing a carriage return without a line feed, and since essentially all of the software habitually sends a line feed after a carriage return, having the decoder software ignore the line feed neatly solves one problem. Additionally, the printer must pause when returning, or else the next character will end up smeared across the page as the typing element flies back to the left margin. Although a set of contacts is available inside the printer to send a signal when the carrier is set at the left margin, interfacing them was felt to be more difficult than providing a simple short delay. Thus a software delay was incorporated in order to provide the pause that refreshes after a carriage return.

Tabbing? Sure, why not. The standard ASCII TAB character, control-I, hex $09, is decoded to the Selectric TAB character. Send a tab out, tab a tab over. What could be simpler?

As noted in the earlier article, several codes are decoded by the Selectric to place the machine on and off line. Since the turn-on code is the number nine, setting the machine up once it is running will cause an extraneous "9" to be printed. I therefore included the turn off code in the table, decoded as the equivalent to ASCII control-S (for Stop), and send this first in the initialization, followed by the turn on character. Thus, if the machine is on line, it is turned off before being turned on again. If it is already off, it, of course, ignores anything until the turn-on code is sent.

Finally, the need for some more intellegence was obvious. So, logic was added to enable the printer driver to keep track of the current line number in a page, and, upon receipt of a Form Feed (control-L or $0C) automatically advance to the first line of the next page. In order to set up the number of lines, an ESCAPE sequence is sent to the printer. Upon receipt of an ESC code ($1B), the program takes the next byte as the page length, and sets the current line counter to zero. If the character following the ESC is $01, however, the page length is not changed, only the line pointer is restored to zero.

When first implemented, the form feed worked well, but, because it used carriage returns to run down the page, it was a bit slow. I then allowed the ASCII vertical tab character (control-K, $0B) to be decoded as the Selectric INDEX character, really a line feed. This allowed the rapid feeding of lines and significantly improved the driver's function.

Other characters that were not represented on the type ball I first used were either assigned as a null or space, as the fancy struck me. At least this allowed me the facility for writing in those omitted characters on drafts requiring one or two of them.

After a while, it became apparent that several of the ASCII character set, most notably the greater and less than signs, were fairly indispensable for BASIC listings and the like. I even liked to use them as attention getters on bills and such. A short search (checking through my office supplier's catalog) turned up the Manifold type ball, which seemed to be just what I needed. This ball has only upper case. but includes most of the ASCII

signs, with a few more thrown in for good measure. Figure 1 is a chart comparing the two balls, in QWERTY order.

It took only a short while to modify the driver program to accomodate the new ball. The checks for period and comma were removed, there is a case difference on the Manifold sphere. And since the Manifold ball has only upper case characters, although they type in lower case positions, the code table was duplicated to output the same code for either ASCII upper or lower case letters.

In order to interface the new driver routines, which I assembled in the $B000 block of memory, to my DOS, several new addresses are inserted into the DOS jump table. Smoke Signal's DOS68 systems include this jump table to tell DOS where everything is. It is a simple matter to give it the addresses of the new initialization and output routines. Once this is done, any program which uses the DOS printer driver addresses will output through the Selectric. Thus, the editor, assembler, or text processor all work with the new system.

The many BASIC interpreters available make it difficult to predict how to interface any particular one. However, most of the Smoke Signal BASICs have a I/O table at the beginning. I patched in the driver addresses as shown in Figure 2, leaving the character input routine tied to my keyboard, which is a PIA on Port 4. Thus, even with the printer selected, should control somehow be transferred there, as with a PORT statement, keyboard input will be maintained. Otherwise, once the printer was assigned as the control port, there would be no way to input data, and the computer would have to be reset to regain control.

At the end of the source listing is a short routine that loads into the transient command area (TCA) of DOS which does several things. First, it begins initialization of the output PIA, which is later finished by the standard initialization routine at $B000. It also prints a banner to tell me which type ball I have selected and what the current page length is. These may not be necessary, I called the two routines GOTHIC.SEL and MNIFLD.SEL so which type ball I get is obvious, but it is nice to know what the computer is doing once in a while.

I was initially afraid at how much kluging I would have to do to get the PIA to drive the RS-232 link. Guess what I found out? The old MP-LA PIA card, with buffers and all, drives the Selectric link directly just fine, accounting for the fact that writing a "1" to the port, which I think of as a mark, drives the port high, which is RS-232 space, and vice versa. So take the programming on the cuff and save some hardware.

How many things can be this simple? A software driver for a Selectric printer that neatly interfaces to DOS and all other major programs. Hardware involved is zip, and the software even adds a few features that the "ASCII Selectrics" don't have. Now, if only a modem was this easy.

Selectric Output For 6800, Leavey

Figure 1

Letter Gothic		MANIFOLD
[ ] @ # $ % ¢ & * ( ) _ +	Upper Case	' " < > # @ ¢ % ¬ | & - /
Q W E R T Y U I O P &		• • • • • • • • • • • •
A S D F G H J K L : "		@ # $ _ ¬ • • • • • •
Z X C V B N M . . ?		& | < ? • • • ; : •
1 2 3 4 5 6 7 8 9 0 - =	Lower Case	1 2 3 4 5 6 7 8 9 0 • "
q w e r t y u i o p &		Q W E R T Y U I O P •
a s d f g h j k l ; '		A S D F G H J K L ( )
z x c v b n m , . /		Z X C V B N M , . •

TYPE BALL COMPARISON

Figure 2

Modification To Port 5

```
0188 04      LDS                 0188 10      LDS
0189 80 14       FDB $8014       0189 80 10       FDB $8010
018B 7E 01FC     JMP CHROUT      018B 7E C320     JMP OUTSEL
018E 7E 0226     JMP CHRIN       018E 7E 0226     JMP CHRIN
0191 7E 01AC     JMP IOINIT      0191 7E B000     JMP INIT
```

Original SS0 BASIC I/O Jumps — Modified BASIC I/O Jumps

Setup to allow a keyboard on Port 4 PIA to provide input to the computer while Selectric printer output is underway.

GOTHIC.SEL — SS0 MNEMONIC ASSEMBLER PAGE 1

```
                    1:       NAM  GOTHIC.SEL
                    2:       OPT  NOS
                    3:       OPT  NOG
                    4: ********************************************
                    5: *   SELECTRIC OUTPUT ROUTINE FOR 6800      *
                    6: *              COMPUTER                    *
                    7: *   VER 1.09 - 4 APR 80 THRU 5 FEB 81      *
                    8: *                BY                        *
                    9: *     MARC I. LEAVEY, M.D.   WA5AJH        *
                   10: ********************************************
                   11: *   LETTER GOTHIC --- BALL NUMBER 126      *
                   12: *              VERSION                     *
                   13: ********************************************
                   14: *
                   15: * DOS JUMPS AND EQUATES
                   16: *
0319               17: PPPORT EQU  $0319
0312               18: ZNCOUT EQU  $0312
030F               19: ZNEINT EQU  $030F
0203               20: ZMAPPR EQU  $0203
E07E               21: POATA  EQU  $E07E
8012               22: PORT   EQU  $8012
009C               23: UPSHFT EQU  $9C
001F               24: DNSHFT EQU  $1F
0010               25: ESC    EQU  $10
```

```
                    ;**********************************
                    ;*      INITIALIZATION ROUTINE    *
                    ;**********************************
0000                        ORG   $8000
8000 7F 8013 INIT   CLR   PORT+1      SET UP PIA
8003 7F 8012        CLR   PORT        FOR OUTPUT
8006 86 FF          LDA A #$FF
8008 B7 8012        STA A PORT
800B 86 04          LDA A #$4
800D B7 8013        STA A PORT+1
8010 86 13          LDA A #$13        TAKE PRINTER OFF LINE
8012 BD 0C          BSR   OUTSEL
8014 86 39          LDA A #$39        PUT BACK ON LINE
8016 BD 08          BSR   OUTSEL
8018 39             RTS               THEN EXIT
                    ;**********************************
                    ;*    CHARACTER OUTPUT ROUTINE    *
                    ;**********************************
8020                        ORG   $8020
8020 7D B12A OUTSEL TST   ESCFLG      IS ESCAPE SEQUENCE IN PROGRESS?
8023 26 08          BNE   FUNCTN      YES - DO A FUNCTION
8025 B1 1B          CMP A #ESC        NO - IS THIS AN ESCAPE?
8027 26 13          BNE   NOTESC      NOT THAT EITHER - SEND CHARACTER
8029 7C B12A SETFLG INC   ESCFLG     YES! SET THE ESC FLAG AND
802C 39             RTS               GO HOME....
                    *** *** ESCAPE SEQUENCE DECODER ***** ***
802D 7F B12A FUNCTN CLR   ESCFLG      CLEAR ESCAPE FLAG
8030 7F B130        CLR   LINECT      CLEAR LINE COUNTER
8033 81 01          CMP A #$01        IS SEQUENCE ESC-1?
8035 27 F5          BEQ   BUGOUT      YEP -- ALL DONE
8037 B7 B131        STA A PAGLEN      NOPE - SET NEW PAGE LENGTH
803A 20 FD          BRA   BUGOUT      THEN GET OUT OF HERE.
                    ;** **** CHARACTER OUTPUT ******
803C B7 B12C NOTESC STA A ASTOR       SAVE ALL INCOMING DATA FOR
803F F7 B120        STA B BSTOR       LATER.
8042 FF B12E        STX   XSTOR
8045 84 7F          AND A #$7F        MASK OFF MSB
8047 B7 B128        STA A ASCCHR      SAVE THE ASCII FOR LATER
804A 81 0D          CMP A #$0D        WAS IT A CAR RET?
804C 26 0E          BNE   NOTCR       NO - TRY ANOTHER CHOICE
804E 7C B130        INC   LINECT      YES - INCREMENT LINE COUNTER
8051 F6 B130        LDA B LINECT      IS THE PAGE
8054 F1 B131        CMP B PAGLEN      FULL YET?
8057 26 03          BNE   NOTCR       NO - GO ON
8059 7F B130        CLR   LINECT      YES - CLEAR LINE COUNTER
805C 81 0C          CMP A #$0C        IS CHARACTER A FORM FEED?
805E 26 17          BNE   NOTFF       NOT THAT EITHER. HUH?
8060 F6 B130        LDA B LINECT      GET CURRENT LINE
8063 F7 B132 FFLOOP STA B LINSAV      AND
8066 86 0B          LDA A #$0B        SEND LINE FEEDS
8068 BD 1E          BSR   NOSHFT      UNTIL
806A F6 B132        LDA B LINSAV      PAGE IS
806D 5C             INC B             DONE
806E F1 B131        CMP B PAGLEN
8071 26 F0          BNE   FFLOOP
8073 7F B130        CLR   LINECT      THEN CLEAR EVERYTHING
8076 4F             CLR A             AND GO ON.
8077 81 20          CMP A #$20        IS IT A SPACE?
8079 2F 00          BLE   NOSHFT
807B 81 2E          CMP A #'.         IS IT A PERIOD?
807D 27 08          BEQ   NOSHFT
807F 81 2C          CMP A #','        IS IT A COMMA?
8081 27 05          BEQ   NOSHFT      IF ANY - SETUP SHIFT BYPASS
8083 7F B129        CLR   PASSFT      OTHERWISE CLEAR THE FLAG
8086 20 06          BRA   GETCHR
8088 7F B128 NOSHFT CLR   PASSFT      SET UP SHIFT BYPASS BY
808B 7C B128        INC   PASSFT      PUTTING A 1 IN PASS SHIFT FLAG
808E B7 8095 GETCHR STA A INDEX+1     PUT CHARACTER I TO TABLE INDEX
8091 CE B133        LDX   #TABLE      SET UP TABLE
8094 A6 00          LDA A 0,X         AND RETRIEVE CHARACTER.
8096 81 7F          CMP A #$7F        IF IT IS $7F THEN
8098 27 4A          BEQ   EXIT        FORGET IT.
809A 7D B128 NOTSEL TST   PASSFT      SHOULD WE WORRY ABOUT SHIFT?
809D 26 29          BNE   SHIFTD      NO - FLAG SET!
809F 40             TST A             IF MSB SET THEN CHAR IS LOWER CASE
80A0 2B 14          BMI   LOCASE
80A2 7D B129 UPCASE TST   SFTFLG      NOW, THE CHARACTER IS UPPER CASE
80A5 26 21          BNE   SHIFTS      IF SHIFT FLAG=1 THEN ALL IS OK
80A7 B7 B127        STA A SELCHR      OTHERWISE SAVE CHARACTER
80AA 86 9C          LDA A #UPSHFT     SEND OUT A SHIFT UP
80AC 8D 4D          BSR   OUTPUT
80AE 7C B129        INC   SFTFLG      SET FLAG
80B1 B6 B127        LDA A SELCHR      RETRIEVE CHARACTER
80B4 20 12          BRA   SHIFTD      AND PUT IT OUT
80B6 7D B129 LOCASE TST   SFTFLG      NOW, THE CHARACTER IS LOW B CASE
80B9 27 0D          BEQ   SHIFTD      IF SHIFT FLAG=0 THEN ALL IS OK
80BB B7 B127        STA A SELCHR      OTHERWISE SAVE CHARACTER
80BE 86 1F          LDA A #UNSHFT     SEND OUT A SHIFT DOWN
80C0 8D 2C          BSR   OUTPUT
80C2 7F B129        CLR   SFTFLG      CLEAR FLAG
80C5 B6 B127        LDA A SELCHR      RETRIEVE CHARACTER
80C8 8D 24          BSR   OUTPUT      AND PUT OUT THE CHARACTER.
80CA B6 B128        LDA A ASCCHR      BEFORE WE LEAVE, GET BACK THE ASCII
80CD 81 0D          CMP A #$0D        WAS IT A CAR RET?
80CF 26 09          BNE   PASSCR      NO - TRY AGAIN
80D1 CE FFFF        LDX   #$FFFF      YES - DELAY FOR A BIT
80D4 09             DEX               TO ALLOW THE CARRIAGE TO REST
80D5 26 FD          BNE   CRL'OP      CRL'OP
80D7 09             DEX               CRLUP2
80D8 26 FD          BNE   CRLUP2      * OMIT THESE TWO LINES FOR
80DA 81 0C          CMP A #$0C        * 1 MHZ OPERATION
80DC 26 06          BNE   EXIT        WAS IT A FORM FEED?
80DE CE 5000        LDX   #$5000      NO - GO HOME NOW
80E1 09             DEX               YES - DELAY FOR A SHORTER TIME
80E2 26 FD          BNE   FFDLUP      $5000 AT 2 MHZ, $2800 AT 1 MHZ
80E4 B6 B12C EXIT   LDA A ASTOR       GET BACK ALL YOU STORED
80E7 F6 B120        LDA B BSTOR       AT THE BEGINNING THEN
80EA FE B12E        LDX   XSTOR       LEAVE
80ED 39             RTS               ... QUIETLY!
                    ;***** SOFTWARE UART ROUTINE *******
80EE C6 07          LDA B #$7         SET UP SEVEN BIT COUNTER
80F0 B7 B126        STA A STOREA
80F3 86 01          LDA A #$1         SEND START BIT
80F5 B7 8012        STA A PORT
80F8 8D 25          BSR   DELAY
80FA 0C             CLC
80FB B6 B126        LDA A STOREA
80FE 47             ASR A             SHIFT OUT EACH DATA BIT
80FF B7 B126        STA A STOREA
8102 24 09          BCC   SEND0
8104 86 00          LDA A #$0         SEND MARK = RS-232 LOW = 0
8106 B7 8012        STA A PORT
8109 8D 14          BSR   DELAY
810B 20 07          BRA   ENDBIT
810D 86 01          LDA A #$1         SEND SPACE = RS-232 HIGH = 1
810F B7 8012        STA A PORT
8112 8D 0B          BSR   DELAY
8114 5A             DEC B             DEC BIT COUNTER
8115 26 E3          BNE   OUTLUP      NOT DONE - DO ANOTHER
8117 86 00          LDA A #$0         DONE - SEND STOP BIT
8119 B7 8012        STA A PORT        AND
811C 8D 01          BSR   DELAY       THEN
811E 39             RTS               RETURN
                    ;***** DELAY LOOP ******
811F CE 073E DELAY  LDX   #1854       * DELAY CONSTANT = 927 AT 1 MHZ, 1854 AT 2 MHZ
                    *  FOR 134.5 BAUD SELECTRIC TRANSMISSION RATE
8122 09             DEX
8123 26 FD          BNE   DLOOP
8125 39             RTS
                    ;***** STORAGE FOR VARIABLES *****
8126                STOREA RMB  1
8127                SELCHR RMB  1
8128                ASCCHR RMB  1
8129 00             SFTFLG FCB  0
812A 00             ESCFLG FCB  0
812B 00             PASSFT FCB  0
812C                ASTOR  RMB  1
812D                BSTOR  RMB  1
812E                XSTOR  RMB  2
8130 00             LINECT FCB  0
8131 42             PAGLEN FCB  66
8132                LINSAV RMB  1
                    ;***** GOTHIC BALL TABLE *****
8133 7F             I 0: TABLE  FCB  $7F              NULL
8134 7F                  FCB  $7F,$7F,$7F,$7F,$7F
8139 7F                  FCB  $7F,$7F
813B 50                  FCB  $50             BACKSPACE
813C 2F                  FCB  $2F             TAB
813D 7F                  FCB  $7F             LINE FEED BLANKED
813E 6E                  FCB  $6E             CTL-K=VERTICAL TAB
813F 7F                  FCB  $7F             CTL-L=FORM FEED
8140 60                  FCB  $60             CAR RET/LINE FEED
8141 7F                  FCB  $7F,$7F,$7F,$7F,$7F
8146 FC                  FCB  $FC             CTRL-S = TURN OFF
8147 7F                  FCB  $7F,$7F,$7F,$7F,$7F
814C 7F                  FCB  $7F,$7F,$7F,$7F,$7F
8151 7F                  FCB  $7F,$7F
8153 40                  FCB  $40             SPACE
8154 20                  FCB  $20             !
8155 49                  FCB  $49             "
8156 70                  FCB  $70             #
8157 04                  FCB  $04             $
8158 08                  FCB  $08             %
8159 68                  FCB  $68             &
815A C9                  FCB  $C9             '
815B 34                  FCB  $34             (
815C 64                  FCB  $64             )
815D 38                  FCB  $38             *
815E 13                  FCB  $13             +
815F 38                  FCB  $38             ,
8160 07                  FCB  $07             -
8161 51                  FCB  $51             .
8162 87                  FCB  $87             /
8163 E4                  FCB  $E4             0
8164 A0                  FCB  $A0             1
8165 90                  FCB  $90             2
8166 F0                  FCB  $F0             3
8167 84                  FCB  $34             4
8168 88                  FCB  $88             5
8169 08                  FCB  $08             6
816A E8                  FCB  $E8             7
816B 88                  FCB  $88             8
816C 84                  FCB  $84             9
816D 68                  FCB  $68             :
816E ED                  FCB  $ED             ;
816F 7F                  FCB  $7F             <
8170 93                  FCB  $93             =
8171 7F                  FCB  $7F             >
8172 07                  FCB  $07             ?
8173 10                  FCB  $10             @
8174 79                  FCB  $79             A
8175 76                  FCB  $76             B
8176 7A                  FCB  $7A             C
8177 2A                  FCB  $2A             D
8178 4A                  FCB  $4A             E
8179 73                  FCB  $73             F
817A 23                  FCB  $23             G
817B 26                  FCB  $26             H
817C 19                  FCB  $19             I
8170 43                  FCB  $43             J
817E 1A                  FCB  $1A             K
817F 46                  FCB  $46             L
8180 61                  FCB  $61             M
8181 52                  FCB  $52             N
8182 45                  FCB  $45             O
8183 0B                  FCB  $0B             P
8184 56                  FCB  $58             Q
8185 29                  FCB  $29             R
8186 25                  FCB  $25             S
8187 02                  FCB  $02             T
8188 32                  FCB  $32             U
8189 31                  FCB  $31             V
818A 75                  FCB  $75             W
818B 62                  FCB  $62             X
818C 67                  FCB  $67             Y
818D 54                  FCB  $54             Z
818E 7F                  FCB  $7F
818F 7F                  FCB  $7F
8190 7F                  FCB  $7F
8191 7F                  FCB  $7F
8192 37                  FCB  $37
                    ;***** LOWER CASE ASCII STARTS HERE *****
8193 7F                  FCB  $7F
8194 F9                  FCB  $F9             a
8195 F6                  FCB  $F6             b
8196 FA                  FCB  $FA             c
8197 AA                  FCB  $AA             d
8198 CA                  FCB  $CA             e
8199 F3                  FCB  $F3             f
819A A3                  FCB  $A3             g
819B A6                  FCB  $A6             h
819C 99                  FCB  $99             i
819D C3                  FCB  $C3             j
819E 9A                  FCB  $9A             k
819F C6                  FCB  $C6             l
81A0 E1                  FCB  $E1             m
81A1 D2                  FCB  $D2             n
81A2 C5                  FCB  $C5             o
81A3 8B                  FCB  $8B             p
81A4 0B                  FCB  $0B             q
81A5 A9                  FCB  $A9             r
81A6 A5                  FCB  $A5             s
81A7 82                  FCB  $82             t
```

```
B1A8 52          280:         FCB   $82          u
B1A9 B1          281:         FCB   $B1          v
B1AA F5          282:         FCB   $F5          w
B1AB E2          283:         FCB   $E2          x
B1AC E7          284:         FCB   $E7          y
B1AD D4          285:         FCB   $D4          z
B1AE 7F          286:         FCB   $7F,$7F,$7F,$7F,$7F
                 287: ********************************************
                 288: *      VECTORS FOR 305680 JUMP TABLE       *
                 289: ********************************************
0319             290:         ORG   VPPORT
0319 80 12       291:         FDB   PORT
                 292: *
D30F             293:         ORG   ZMCINT
D30F 7E B000     294:         JMP   INIT
                 295: *
D312             296:         ORG   ZMCOUT
D312 7E B020     297:         JMP   OUTSEL
                 298: ********************************************
                 299: * TRANSFER ADDRESS ENTERS HERE FOR BANNER *
                 300: *          NO USER INPUT REQUESTED         *
                 301: ********************************************
CD8D             302:         ORG   $CD8D
CD8D CE CE32     303: PROMPT  LDX   #LINPNT     PUT OUT BANNER
CD90 BD E07E     304:         JSR   PDATA
CD93 7F 8012     305:         CLR   PORT
CD96 7F 8013     306:         CLR   PORT+1
CD99 7F 8012     307:         CLR   PORT
CD9C 7E D283     308:         JMP   ZWARMS     AND RETURN TO DOS
                 309: *
CE32 20          310: LINPNT  FCB   $D,$A,$A
CE35 53          311:         FCC   /Selectric input routine initialized at 66 line page/
CE67 0D          312:         FCB   $D,$A
CE69 4C          313:         FCC   /Letter Gothic Type Ball/
CE8D 04          314:         FCB   4
                 315: ********************************************
CE8D             316:         END   PROMPT
    NO ERROR(S) DETECTED

                   1:         NAM   PRINTER.SEL
                   2:         OPT   NOS
                   3:         OPT   MOS
                   4: ********************************************
                   5: *   SELECTRIC OUTPUT ROUTINE FOR 6800      *
                   6: *              COMPUTER                    *
                   7: *   VER 1.10 - 8 APR 80 THRU 5 FEB 81      *
                   8: *              BY                          *
                   9: *   MARC I. LEAVEY, M.D.    WA3AJR       *
                  10: ********************************************
                  11: *   MANIFOLD 72 --- BALL NUMBER 087        *
                  12: *              VERSION                     *
                  13: ********************************************
                  14: *
                  15: * DOS JUMPS AND EQUATES
                  16: *
0319              17: VPPORT EQU   $0319
D312              18: ZMCOUT EQU   $D312
D30F              19: ZMCINT EQU   $D30F
D283              20: ZWARMS EQU   $D283
E07E              21: PDATA  EQU   $E07E
8012              22: PORT   EQU   $8012
009C              23: UPSHFT EQU   $9C
001F              24: DWSHFT EQU   $1F
001B              25: ESC    EQU   $1B
                  26: ********************************************
                  27: *          INITIALIZATION ROUTINE          *
                  28: ********************************************
8000              29:         ORG   $8000
8000 7F 8013      30: INIT    CLR   PORT+1     SET UP PIA
8003 7F 8012      31:         CLR   PORT       FOR OUTPUT
8006 86 FF        32:         LDA A #$FF
8008 B7 8012      33:         STA A PORT
800B 86 04        34:         LDA A #$4
800D B7 8013      35:         STA A PORT+1
8010 86 13        36:         LDA A #$13       TAKE PRINTER OFF LINE
8012 BD 0C        37:         BSR   OUTSEL
8014 86 39        38:         LDA A #$39       PUT BACK ON LINE
8016 BD 08        39:         BSR   OUTSEL
8018 39           40:         RTS              THEN EXIT
                  41: ********************************************
                  42: *        CHARACTER OUTPUT ROUTINE          *
                  43: ********************************************
8020              44:         ORG   $8020
8020 7D 8122      45: OUTSEL  TST   ESCFLG     IS ESCAPE SEQUENCE IN PROGRESS?
8023 26 08        46:         BNE   FUNCTN     YES - DO A FUNCTION
8025 81 1B        47:         CMP A #ESC       NO - IS THIS AN ESCAPE?
8027 26 13        48:         BNE   NOTESC     NOT THAT EITHER - SEND CHARACTER
8029 7C 8122      49: SETFLG  INC   ESCFLG     YES! SET THE ESC FLAG AND
802C 39           50: BLKOUT  RTS              GO HOME...
                  51:         ------ ESCAPE SEQUENCE DECODED ------
802D 7F 8122      52: FUNCTN  CLR   ESCFLG     CLEAR ESCAPE FLAG
8030 7F 8128      53:         CLR   LINECT     CLEAR LINE COUNTER
8033 81 01        54:         CMP A #$01       IS SEQUENCE ESC-1?
8035 27 F5        55:         BEQ   BLKOUT     YEP -- ALL DONE
8037 B7 8129      56:         STA A PAGLEN     NOPE - SET NEW PAGE LENGTH
803A 20 F0        57:         BRA   BLKOUT     THEN GET OUT OF HERE.
                  58:         ------ CHARACTER OUTPUT ROUTINE ------
803C B7 8124      59: NOTESC  STA A ASTOR      SAVE ALL INCOMING DATA FOR
803F F7 8125      60:         STA B BSTOR      LATER.
8042 FF 8126      61:         STX   XSTOR
8045 84 7F        62:         AND A #$7F       MASK OFF MSB
8047 B7 8120      63:         STA A ASCCHR     SAVE THE ASCII FOR LATER
804A 81 0D        64:         CMP A #$0D       WAS IT A CAR RET?
804C 26 0E        65:         BNE   NOTCR      NO - TRY ANOTHER CHOICE
804E 7C 8128      66:         INC   LINECT     YES - INCREMENT LINE COUNTER
8051 F6 8128      67:         LDA B LINECT     IS THE PAGE
8054 F1 8129      68:         CMP B PAGLEN     FULL YET?
8057 26 07        69:         BNE   NOTCR      NO - GO ON
8059 7F 8128      70:         CLR   LINECT     YES - CLEAR LINE COUNTER
805C 81 3C        71: NOTCR   CMP A #$0C       IS CHARACTER A FORM FEED?
805E 26 17        72:         BNE   NOTFF      NOT THAT EITHER, HUH?
8060 F6 8128      73:         LDA B LINECT     GET CURRENT LINE
8063 F7 812A      74: FFLOOP  STA B LINSAV     AND
8066 86 0A        75:         LDA A #$0A       SEND LINE FEEDS
8068 BD 16        76:         BSR   NOBSHFT    UNTIL
806A F6 812A      77:         LDA B LINSAV     PAGE IS
806D 5C           78:         INC B            DONE
806E F1 8129      79:         CMP B PAGLEN
8071 26 F0        80:         BNE   FFLOOP
8073 7F 8128      81:         CLR   LINECT     THEN CLEAR EVERYTHING
8076 4F           82:         CLR A            AND GO ON.
8077 81 20        83: NOTFF   CMP A #$20       IS IT A SPACE?
8079 2F 05        84:         BLE   NOBSHFT    SPACE IS THE ONLY CHARACTER ON THE
                  85:         ------ MANIFOLD BALL IN BOTH UPPER AND LOWER
                  86:         ------ CASE
807B 7F 8123      87:         CLR   PASSFT     OTHERWISE CLEAR THE FLAG
807E 20 06        88:         BRA   GETCHR

8083 7F 8123      89: NOBSHFT CLR   PASSFT     SET UP SHIFT BYPASS BY
8086 7C 8123      90:         INC   PASSFT     PUTTING A 1 IN PASS SHIFT FLAG
8089 B7 8360      91: GETCHR  STA A INDEX+1    PUT CHARACTER INTO TABLE INDEX
808C CE 8128      92:         LDX   #TABLE     SET UP TABLE
808F A6 00        93: INDEX   LDA A 0,X        AND RETRIEVE CHARACTER.
8091 81 7F        94:         CMP A #$7F       IF IT IS $7F THEN
8093 27 4A        95:         BEQ   EXIT       FORGET IT.
8095 7D 8123      96: BOTSEL  TST   PASSFT     SHOULD WE WORRY ABOUT SHIFT?
8098 26 29        97:         BNE   SHIFTD     NO - FLAG SET!
809A 4D           98:         TST A            IF MSB SET THEN CHAR IS LOWER CASE
809B 2B 19        99:         BMI   LOCASE
809D 7D 8121     100: UPCASE  TST   SFTFLG     NOW, THE CHARACTER IS UPPER CASE
80A0 26 21       101:         BNE   SHIFTD     IF SHIFT FLAG=1 THEN ALL IS OK
80A2 B7 811F     102:         STA A SELCHR     OTHERWISE SAVE CHARACTER
80A5 86 5C       103:         LDA A #UPSHFT    SEND OUT A SHIFT UP
80A7 BD 40       104:         BSR   OUTPUT
80A9 7C 8121     105:         INC   SFTFLG     SET FLAG
80AC B6 811F     106:         LDA A SELCHR     RETRIEVE CHARACTER
80AF 20 12       107:         BRA   SHIFTD     AND PUT IT OUT
80B1 7D 8121     108: LOCASE  TST   SFTFLG     NOW, THE CHARACTER IS LOWER CASE
80B4 27 0D       109:         BEQ   SHIFTD     IF SHIFT FLAG=0 THEN ALL IS OK
80B6 B7 811F     110:         STA A SELCHR     OTHERWISE SAVE CHARACTER
80B9 86 1F       111:         LDA A #DWSHFT    SEND OUT A SHIFT DOWN
80BB BD 2C       112:         BSR   OUTPUT
80BD 7F 8121     113:         CLR   SFTFLG     CLEAR FLAG
80C0 B6 811F     114:         LDA A SELCHR     RETRIEVE CHARACTER
80C3 BD 30       115: SHIFTD  BSR   OUTPUT     AND PUT OUT THE CHARACTER.
80C5 B6 8120     116:         LDA A ASCCHR     BEFORE WE LEAVE, GET BACK THE ASCII
80C8 31 0D       117:         CMP A #$0D       WAS IT A CAR RET?
80CA 26 09       118:         BNE   PASSCR     NO - TRY AGAIN
80CC CE FFFF     119: CRLDLAY LDX   #$FFFF     YES - DELAY FOR A BIT
80CF 09          120: CRLOOP  DEX              TO ALLOW THE CARRIAGE TO REST
80D0 26 FD       121:         BNE   CRLOOP
80D2 09          122: CRLUP2  DEX              * OMIT THESE TWO LINES FOR
80D3 26 FD       123:         BNE   CRLUP2     * 1 MHZ OPERATION
80D5 81 0C       124: PASSCR  CMP A #$0C       WAS IT A FORM FEED?
80D7 26 06       125:         BNE   EXIT       NO - GO HOME NOW
80D9 CE 5000     126: FFDLAY  LDX   #$5000     YES - DELAY FOR A SHORTER TIME
80DC 09          127: FFDLUP  DEX              $5000 AT 2 MHZ, $2800 AT 1 MHZ
80DD 26 FD       128:         BNE   FFDLUP
80DF B6 8124     129: EXIT    LDA A ASTOR      GET BACK ALL YOU STORED
80E2 F6 8125     130:         LDA B BSTOR      AT THE BEGINNING THEN
80E5 FE 8126     131:         LDX   XSTOR      LEAVE
80E8 39          132:         RTS              ... OUTPUT
                 133:         ------ SOFTWARE UART ROUTINE ------
80E9 C6 07       134: OUTPUT  LDA B #7         SET UP SEVEN BIT COUNTER
80EB B7 811E     135:         STA A STOREA
80EE 86 01       136:         LDA A #1         SEND START BIT
80F0 B7 8012     137:         STA A PORT
80F3 BD 25       138:         BSR   DELAY
80F5 0C          139: OUTLUP  CLC
80F6 B6 811E     140:         LDA A STOREA
80F9 47          141:         ASR A            SHIFT OUT EACH DATA BIT
80FA B7 811E     142:         STA A STOREA
80FD 24 09       143:         BCC   SENDO
80FF 86 00       144: SEND1   LDA A #$0        SEND MARK = RS-232 LOW -> 0
8101 B7 8012     145:         STA A PORT
8104 BD 1A       146:         BSR   DELAY
8106 20 07       147:         BRA   ENDBIT
8108 86 01       148: SENDO   LDA A #1         SEND SPACE = RS-232 HIGH -> 1
810A B7 8012     149:         STA A PORT
810D BD 0B       150:         BSR   DELAY
810F 5A          151: ENDBIT  DEC B            DEC BIT COUNTER
8110 26 E3       152:         BNE   OUTLUP     NOT DONE - DO ANOTHER
8112 86 00       153:         LDA A #$0        DONE - SEND STOP BIT
8114 B7 8012     154:         STA A PORT       AND
8117 BD 01       155:         BSR   DELAY      THEN
8119 39          156:         RTS              RETURN!
                 157:         ------ DELAY LOOP ------
811A CE 073E     158: DELAY   LDX   #$83A      * DELAY CONSTANT = 927 AT 1 MHZ, 1853 AT 2 MHZ
                 159:         * FOR 134.5 BAUD SELECTRIC TRANSMISSION RATE
811D 09          160: DLOOP   DEX
811E 26 FD       161:         BNE   DLOOP
8120 39          162:         RTS
                 163:         ------ STORAGE FOR VARIABLES ------
8121             164: STOREA  RMB   1
8122             165: SELCHR  RMB   1
8123             166: ASCCHR  RMB   1
8124 00          167: SFTFLG  FCB   0
8125 00          168: ESCFLG  FCB   0
8126 00          169: PASSFT  FCB   0
8127             170: ASTOR   RMB   1
8128             171: BSTOR   RMB   1
8129             172: XSTOR   RMB   2
812B 00          173: LINECT  FCB   0
812C 42          174: PAGLEN  FCB   66
812D             175: LINSAV  RMB   1
                 176:         ------ MANIFOLD BALL TABLE ------
812E 7F          177: TABLE   FCB   $7F          NULL
812F 7F          178:         FCB   $7F,$7F,$7F,$7F,$7F
8134 7F          179:         FCB   $7F,$7F
8136 5D          180:         FCB   $5D          BACKSPACE
8137 2F          181:         FCB   $2F          TAB
8138 7F          182:         FCB   $7F          LINE FEED BLANKED
8139 6E          183:         FCB   $6E          CTL-N=VERTICAL TAB
813A 7F          184:         FCB   $7F          CTL-L=FORM FEED
813B 60          185:         FCB   $60          CAR RET/LINE FEED
813C 7F          186:         FCB   $7F,$7F,$7F,$7F,$7F
8141 FC          187:         FCB   $FC          CTRL-S = TURN OFF
8142 7F          188:         FCB   $7F,$7F,$7F,$7F,$7F
8147 7F          189:         FCB   $7F,$7F,$7F,$7F,$7F
814C 7F          190:         FCB   $7F,$7F,$7F,$7F,$7F
8151 62          191:         FCB   $62          SPACE
8152 10          192:         FCB   $10          !
8153 25          193:         FCB   $25          "
8154 2A          194:         FCB   $2A          #
8155 5A          195:         FCB   $5A          $
8156 6A          196:         FCB   $6A          %
8157 20          197:         FCB   $20          &
8158 EB          198:         FCB   $EB          '
8159 C9          199:         FCB   $C9          (
815A 93          200:         FCB   $93          )
815B B7          201:         FCB   $B7          *
815C BB          202:         FCB   $BB          +
815D 37          203:         FCB   $37          ,
815E D1          204:         FCB   $D1          -
815F 13          205:         FCB   $13          .
8160 E4          206:         FCB   $E4          /
8161 A0          207:         FCB   $A0          0
8162 90          208:         FCB   $90          1
8163 F0          209:         FCB   $F0          2
8164 8A          210:         FCB   $8A          3
8165 88          211:         FCB   $88          4
8166 D8          212:         FCB   $D8          5
8167 E8          213:         FCB   $E8          6
8168 B8          214:         FCB   $B8          7
8169 B4          215:         FCB   $B4          8
816A             216:         FCB   $8A          9
```

```
0165 51          219:        FCB   $51          :
0166 38          220:        FCB   $38          ;
0167 70          221:        FCB   $70          <
0168 81          222:        FCB   $81          =
0169 68          223:        FCB   $68          >
016A 31          224:        FCB   $31          ?
016B 79          225:        FCB   $79          @
016C F9          226:        FCB   $F9          A
016D F6          227:        FCB   $F6          B
016E FA          228:        FCB   $FA          C
016F AA          229:        FCB   $AA          D
0170 CA          230:        FCB   $CA          E
0171 F3          231:        FCB   $F3          F
0172 A3          232:        FCB   $A3          G
0173 A6          233:        FCB   $A6          H
0174 99          234:        FCB   $99          I
0175 C3          235:        FCB   $C3          J
0176 9A          236:        FCB   $9A          K
0177 C6          237:        FCB   $C6          L
0178 E1          238:        FCB   $E1          M
0179 D2          239:        FCB   $D2          N
017A C5          240:        FCB   $C5          O
017B B8          241:        FCB   $B8          P
017C D8          242:        FCB   $D8          Q
017D A9          243:        FCB   $A9          R
017E A5          244:        FCB   $A5          S
017F B2          245:        FCB   $B2          T
0180 B2          246:        FCB   $B2          U
0181 B1          247:        FCB   $B1          V
0182 F5          248:        FCB   $F5          W
0183 E2          249:        FCB   $E2          X
0184 E7          250:        FCB   $E7          Y
0185 DA          251:        FCB   $DA          Z
0186 7F          252:        FCB   $7F
0187 7F          253:        FCB   $7F
0188 7F          254:        FCB   $7F
0189 08          255:        FCB   $08          ←
018A 73          256:        FCB   $73
                 257:  ******* LOWER CASE ASCII STARTS HERE *******
018B 7F          258:        FCB   $7F
018C F9          259:        FCB   $F9          A
018D F6          260:        FCB   $F6          B
018E FA          261:        FCB   $FA          C
018F AA          262:        FCB   $AA          D
0190 CA          263:        FCB   $CA          E
0191 F3          264:        FCB   $F3          F
0192 A3          265:        FCB   $A3          G
0193 A6          266:        FCB   $A6          H
0194 99          267:        FCB   $99          I
0195 C3          268:        FCB   $C3          J
0196 9A          269:        FCB   $9A          K
0197 C6          270:        FCB   $C6          L
0198 E1          271:        FCB   $E1          M
0199 D2          272:        FCB   $D2          N
019A C5          273:        FCB   $C5          O
019B B8          274:        FCB   $B8          P
019C D8          275:        FCB   $D8          Q
019D A9          276:        FCB   $A9          R
019E A5          277:        FCB   $A5          S
019F B2          278:        FCB   $B2          T
01A0 B2          279:        FCB   $B2          U
01A1 B1          280:        FCB   $B1          V
01A2 F5          281:        FCB   $F5          W
01A3 E2          282:        FCB   $E2          X
01A4 E7          283:        FCB   $E7          Y
01A5 DA          284:        FCB   $DA          Z
01A6 7F          285:        FCB   $7F,$7F,$7F,$7F,$7F
                 286: ********************************************
                 287: *   VECTORS FOR DOS68 JUMP TABLE           *
                 288: ********************************************
0319             289:        ORG   YPPORT
0319 80 12       290:        FDB   PORT
                 291: *
031F             292:        ORG   Z+CINT
031F 7E 8000     293:        JMP   $INIT
                 294: *
0312             295:        ORG   Z+BOUT
0312 7E 8020     296:        JMP   OUTSEL
                 297: ********************************************
                 298: *  TRANSFER ADDRESS ENTERS HERE FOR BANNER  *
                 299: *        IO USER INPUT REQUESTED           *
                 300: ********************************************
CD80             301:        ORG   $CD80
CD80 CE CD92     302: PROMPT LDX   #LINPRT      PUT OUT BANNER
CD83 BD E07E     303:        JSR   PDATA
CD86 7F 8012     304:        CLR   PORT         CLEAR PIA PORT
CD89 7F 8013     305:        CLR   PORT+1
CD8C 7F 8012     306:        CLR   PORT
CD8F 7E DA83     307:        JMP   ZMAIN        AND RETURN TO DOS
                 308: *
CD92 0D          309: LINPRT FCB   $D,$A,$A
CD95 53          310:        FCC   /SELECTRIC INPUT ROUTINE INITALIZED AT 66 LINE PAGE/
CDC7 0D          311:        FCB   $D,$A
CDC9 4D          312:        FCC   /MAINFOLD TYPE BALL/
CDD8 04          313:        FCB   4
                 314: ********************************************
CD80             315:        END   PROMPT
     NO ERROR(S) DETECTED
```

HEAD CLEAN - KEY LOCK

The following two programs are being submitted for public use. The first one is for cleaning the heads of 5" drives. It is to be used with one of the cleaning disks on the market. The second program, written by a friend of mine, is very useful if you have small children or animals around the house that like to hit keys on the terminal while you are away from the computer, say to answer the phone or the door. It will lock up the computer until you enter the correct code. It's also useful to include in a startup routine so the system can't finish booting until it gets the right code. It can also be used with any program that allows you to send commands to Flex such as Basic.

I have been getting Micro 68 since Volume 1, Issue 1 and I treat them like gold. They are invaluable as a reference source that I use at least once a week. Keep up the good work!

Joseph Aulicino
2014 - 59 th Street
Bklyn, N.Y. 11204

P.S. The Flex9.equ file contains all Flex equates used with the LIB function of the assembler.

```
 1.00=*
 2.00=* THIS IS THE STANDARD FLEX09 LABELS
 3.00=*
 4.00=*
 5.00=*
 6.00=*MONITOR EQUATES
 7.00=*
 8.00=ACIA   EQU    $E004
 9.00=*
10.00=*FLEX SYSTEM RAM VARIABLES
11.00=*
12.00=LINBUF  EQU    $C080
13.00=TTYBS   EQU    $CC00
14.00=TTYDEL  EQU    $CC01
15.00=TTYEOL  EQU    $CC02
16.00=TTYDP   EQU    $CC03
17.00=TTYWD   EQU    $CC04
18.00=TTYML   EQU    $CC05
19.00=TTYTB   EQU    $CC06
20.00=TTYBE   EQU    $CC07
21.00=TTYEJ   EQU    $CC08
22.00=TTYPS   EQU    $CC09
23.00=TTYESC  EQU    $CC0A
24.00=SYSDRV  EQU    $CC0B
25.00=WRKDRV  EQU    $CC0C
26.00=SYSMTH  EQU    $CC0E
27.00=SYSDAY  EQU    $CC0F
28.00=SYSYR   EQU    $CC10
29.00=LSTTRK  EQU    $CC11
30.00=UCMDTB  EQU    $CC12
31.00=LINPTR  EQU    $CC14
32.00=ESCRET  EQU    $CC16
33.00=CURCHR  EQU    $CC18
34.00=PRVCHR  EQU    $CC19
35.00=CURLIN  EQU    $CC1A
36.00=LDROFF  EQU    $CC1B
37.00=IFERFG  EQU    $CC1D
38.00=IFERAD  EQU    $CC1E
39.00=ERRTYP  EQU    $CC20
40.00=SPECIO  EQU    $CC21
41.00=OUTSW   EQU    $CC22
42.00=INSW    EQU    $CC23
43.00=OUTFIL  EQU    $CC24
44.00=INFIL   EQU    $CC26
45.00=CRDFLG  EQU    $CC28
46.00=CURCLR  EQU    $CC29
47.00=RETREND EQU    $CC2B
48.00=ERRVEC  EQU    $CC2D
49.00=FILEKO  EQU    $CC2F
50.00=CPUTYP  EQU    $CC33
51.00=PRTADR  EQU    $CC35
52.00=PRTLNG  EQU    $CC37
53.00=PRTDVC  EQU    $CC39
54.00=PINIT   EQU    $CCC0
55.00=PCHR    EQU    $CCD0
56.00=POUT    EQU    $CCE4
57.00=*
58.00=*FLEX USER-CALLABLE ROUTINES
59.00=*
60.00=COLDS   EQU    $CD00
61.00=WARMS   EQU    $CD03
62.00=RENTER  EQU    $CD06
63.00=INCH    EQU    $CD09
64.00=INCH2   EQU    $CD0C
65.00=OUTCH   EQU    $CD0F
66.00=OUTCH2  EQU    $CD12
67.00=GETCHR  EQU    $CD15
68.00=PUTCHR  EQU    $CD18
69.00=INBUFF  EQU    $CD1B
70.00=PSTRNG  EQU    $CD1E
71.00=CLASS   EQU    $CD21
72.00=PCRLF   EQU    $CD24
73.00=NITCH   EQU    $CD27
74.00=RSTRIO  EQU    $CD2A
75.00=GETFIL  EQU    $CD2D
76.00=LOAD    EQU    $CD30
77.00=SETEXT  EQU    $CD33
78.00=ADDBX   EQU    $CD36
79.00=OUTDEC  EQU    $CD39
80.00=OUTHEX  EQU    $CD3C
81.00=RPTERR  EQU    $CD3F
82.00=GETHEX  EQU    $CD42
83.00=OUTADR  EQU    $CD45
84.00=INDEC   EQU    $CD48
85.00=DOCMND  EQU    $CD4B
86.00=STAT    EQU    $CD4E
87.00=*
88.00=*FMS ADDRESSES
89.00=*
90.00=SYSFCB  EQU    $CB40
91.00=FMSINT  EQU    $D400
92.00=FMSCLS  EQU    $D403
93.00=FMS     EQU    $D406
94.00=BASFCB  EQU    $D409
95.00=CURFCB  EQU    $B40B
96.00=VRFYFB  EQU    $B435
97.00=*
98.00= END
```

```
**********************************************
* Command to be used with Cleaning disk, *
* will step heads back and forth for     *
* approx. 30 sec...                      *
**********************************************

*********************************
* Written By Joseph Aulicino *
*********************************

     * Syntax: CLEAN,[Drive #]

     * FLEX EQUATE FILE

     *
     * THIS IS THE STANDARD FLEX09 LABELS
     *
     *
     *
     *MONITOR EQUATES

E004 ACIA    EQU    $E004

     *FLEX SYSTEM RAM VARIABLES

CC80 LINBUF  EQU    $CC80
CC00 TTYBS   EQU    $CC00
CC01 TTYDEL  EQU    $CC01
CC02 TTYEOL  EQU    $CC02
CC03 TTYDP   EQU    $CC03
CC04 TTYNUL  EQU    $CC04
CC05 TTYML   EQU    $CC05
CC06 TTYTB   EQU    $CC06
CC07 TTYBE   EQU    $CC07
CC08 TTYEJ   EQU    $CC08
CC09 TTYPS   EQU    $CC09
CC0A TTYESC  EQU    $CC0A
CC0B SYSDRV  EQU    $CC0B
CC0C WRKDRV  EQU    $CC0C
CC0E SYSMTH  EQU    $CC0E
CC0F SYSDAY  EQU    $CC0F
CC10 SYSYR   EQU    $CC10
CC11 LSTTRM  EQU    $CC11
CC12 UCMDTB  EQU    $CC12
CC14 LINPTR  EQU    $CC14
CC16 ESCRET  EQU    $CC16
CC18 CURCHR  EQU    $CC18
CC19 PRVCHR  EQU    $CC19
CC1A CURLIN  EQU    $CC1A
CC1B LDROFF  EQU    $CC1B
CC1D IFERFG  EQU    $CC1D
CC1E IFERAD  EQU    $CC1E
CC20 ERRTYP  EQU    $CC20
CC21 SPECIO  EQU    $CC21
CC22 OUTSW   EQU    $CC22
CC23 INSW    EQU    $CC23
CC24 OUTFIL  EQU    $CC24
CC26 INFIL   EQU    $CC26
CC28 CMDFLE  EQU    $CC28
CC29 CURCLN  EQU    $CC29
CC2B MEMEND  EQU    $CC2B
CC2D ERRVEC  EQU    $CC2D
CC2F FILEKO  EQU    $CC2F
CC33 CPUTYP  EQU    $CC33
CC35 PRTADR  EQU    $CC35
CC37 PRTLNG  EQU    $CC37
CC39 PRTDVC  EQU    $CC39
CCC0 PINIT   EQU    $CCC0
CCDB PCHK    EQU    $CCDB
CCE4 POUT    EQU    $CCE4

     *FLEX USER-CALLABLE ROUTINES

CD00 COLDS   EQU    $CD00
CD03 WARMS   EQU    $CD03
CD06 RENTER  EQU    $CD06
CD09 INCH    EQU    $CD09
CD0C INCH2   EQU    $CD0C
CD0F OUTCH   EQU    $CD0F
CD12 OUTCH2  EQU    $CD12
CD15 GETCHR  EQU    $CD15
CD18 PUTCHR  EQU    $CD18
CD1B INBUFF  EQU    $CD1B
CD1E PSTRNG  EQU    $CD1E
CD21 CLASS   EQU    $CD21
CD24 PCRLF   EQU    $CD24
CD27 NXTCH   EQU    $CD27
CD2A RSTRIO  EQU    $CD2A
CD2D GETFIL  EQU    $CD2D
CD30 LOAD    EQU    $CD30
CD33 SETEXT  EQU    $CD33
CD36 ADDBX   EQU    $CD36
CD39 OUTDEC  EQU    $CD39
CD3C OUTHEX  EQU    $CD3C
CD3F RPTERR  EQU    $CD3F
CD42 GETHEX  EQU    $CD42
CD45 OUTADR  EQU    $CD45
CD48 INDEC   EQU    $CD48
CD4B DOCMND  EQU    $CD4B
CD4E STAT    EQU    $CD4E

     *FMS ADDRESSES

CB40 SYSFCB  EQU    $CB40
D400 FMSINT  EQU    $D400
0403 FMSCLS  EQU    $0403
0406 FMS     EQU    $0406
0409 BASFCB  EQU    $0409
040B CURFCB  EQU    $040B
0435 VRFYFG  EQU    $0435

          END

     * DC-2 CONTROLLER EQUATES

E014 DRVREG  EQU    $E014    Drive Register
E01B COMREG  EQU    $E01B    Command, Status Register
E019 TRKREG  EQU    $E019    Track Register
E01A SECREG  EQU    $E01A    Sector Register
E01B DATREG  EQU    $E01B    Data Register

C100                 ORG    $C100

C100 20  04     CLEAN  BRA    CLEAN1   branch over variables

C102 01         VN     FCB    1        version number
C103 0000       TEMP   FDB    0        temporary storage
C105 03         COUNT  FCB    3        Cleaning cycles approx. (30 sec.)

                * Program start

C106 BD  CD42   CLEAN1 JSR    GETHEX   Get drive #
C109 25  66             BCS    DRVERR   Error with drive # ?
C10B 8F  C103           STX    TEMP
C10E F6  C104           LDB    TEMP+1   Put drive # in B reg.
C111 20  5E             BRA    DRVERR   Check for valid (0-3)
C113 C1  03             CMPB   #3       drive number
C115 2E  5A             BGT    DRVERR   Error ?
C117 1F  98             TFR    B,A      Construct ASCII character
C119 8B  30             ADDA   #'0      in A reg. For insertion
C11B B7  C184           STA    DRVNUM   in MESS1
C11E 8E  C195           LDX    #MESS1   Insert disk message
C121 BD  CD1E           JSR    PSTRNG   Print message

C124 BD  CD4E   KEYPRS JSR     STAT     Check if Key Pressed
C127 27  FB             BEQ    KEYPRS
C129 7D  E018           TST    COMREG   Start drive motor
C12C F7  E014           STB    DRVREG   of selected drive
C12F BD  CD24           JSR    PCRLF    Print CRLF
C132 8E  C1C7           LDX    #MESS2   Point to running message
C135 BD  CD1E           JSR    PSTRNG   Print message

                * Restore heads

C138 86  0B     RESTOR LDA     #$0B     Restore (tk=00) command
C13A B7  E018          STA     COMREG   Execute command
C13D C6  04            LDB     #4       Delay before checking
C13F 5A         WAIT   DECB             Status register
C140 26  FD            BNE     WAIT
C142 BD  20            BSR     READY    Check for ready
C144 B6  C105          LDA     COUNT    Get cycle count
C147 4A                DECA             Decrement count
C148 B7  C105          STA     COUNT    Store count
```

```
C14B 27  2B          BEQ   EXIT      Is cleaning finished?

                     * Step heads in

C14D 86  5B      STEPIN LDA   #$5B    Step heads in command
C14F B7  E018          STA   COMREG   Execute command
C152 9E  0000          LDX   #0       Delay .5sec
C154 30  1F      DLY   LEAX  -1,X     between head steps
C157 26  FC            BNE   DLY
C159 8D  09            BSR   READY    Check for Ready
C15B B6  E019          LDA   TRKREG   Have we reached
C15E 81  22            CMPA  #$22     maximum Track
C160 26  EB            BNE   STEPIN   yet ?
C162 20  04            BRA   RESTOR   Return for next cycle

                     * Check status

C164 F6  E018    READY LDB   COMREG   Wait for BUSY Status
C167 C5  01            BITB  #1       bit to be cleared
C169 26  F9            BNE   READY
C16B BD  C04E          JSR   STAT     Check for ABORT
C16E 26  0A            BNE   EXIT     Do we abort cleaning ?
C170 39                RTS

                     * Print error messages

C171 8E  C185    DRVERR LDX  #ERRMSG  Point to error message
C174 BD  CD24          JSR   PCRLF    Print CRLF
C177 BD  CD1E          JSR   PSTRNG   Print message

                     * Clean up & Exit

C17A 7F  E018    EXIT  CLR   COMREG   Clear command register
C17D 86  80            LDA   #$80     Deselect drive command
C17F B7  E014          STA   DRVREG   Execute deselect
C182 7E  CD03          JMP   WARMS    Go back to FLEX

                     * Messages

C185 49 6E 76 61 ERRMSG FCC   'Invalid Drive #'
C189 6C 69 64 20
C18D 44 72 69 76
C191 65 20 23
C194 04                FCB   4
C195 49 6E 73 65 MESS1 FCC   'Insert Cleaning Disk in Drive #'
C199 72 74 20 43
C19D 6C 65 61 6E
C1A1 69 6E 67 20
C1A5 44 69 73 6B
C1A9 20 69 6E 20
C1AD 44 72 69 76
C1B1 65 20 23
C1B4 00          DRVNUM FCB   0
C1B5 20 61 6E 64       FCC   ' and Press Return'
C1B9 20 50 72 65
C1BD 73 73 20 52
C1C1 65 74 75 72
C1C5 6E
C1C6 04                FCB   4
C1C7 43 6C 65 61 MESS2 FCC   'Cleaning in Progress - Press '
C1CB 6E 69 6E 67
C1CF 20 69 6E 20
C1D3 50 72 6F 67
C1D7 72 65 73 73
C1DB 20 28 20 50
C1DF 72 65 73 73
C1E3 20
C1E4 52 65 74 75        FCC   'Return to Abort!'
C1E8 72 6E 20 74
C1EC 6F 20 41 62
C1F0 6F 72 74 21
C1F4 04                FCB   4
                       END   CLEAN
```

```
                     *LOCK OUT UNAUTHORIZED USERS

                     *
                     * THIS IS THE STANDARD FLEX09 LABELS
                     *
                     *
                     *
                     *MONITOR EQUATES

E004 ACIA    EQU   $E004

                     *FLEX SYSTEM RAM VARIABLES

CC80 LINBUF  EQU   $CC80
CC00 TTYBS   EQU   $CC00
CC01 TTYDEL  EQU   $CC01
CC02 TTYEOL  EQU   $CC02
CC03 TTYDP   EQU   $CC03
CC04 TTYWD   EQU   $CC04
CC05 TTYNL   EQU   $CC05
CC06 TTYTB   EQU   $CC06
CC07 TTYBE   EQU   $CC07
CC08 TTYEJ   EQU   $CC08
CC09 TTYPS   EQU   $CC09
CC0A TTYESC  EQU   $CC0A
CC0B SYSDRV  EQU   $CC0B
CC0C WRKDRV  EQU   $CC0C
CC0E SYSMTH  EQU   $CC0E
CC0F SYSDAY  EQU   $CC0F
CC10 SYSYR   EQU   $CC10
CC11 LSTTRK  EQU   $CC11
CC12 UCMDTB  EQU   $CC12
CC14 LINPTR  EQU   $CC14
CC16 ESCRET  EQU   $CC16
CC18 CURCHR  EQU   $CC18
CC19 PRVCHR  EQU   $CC19
CC1A CURLIN  EQU   $CC1A
CC1B LDROFF  EQU   $CC1B
CC1D XFERFG  EQU   $CC1D
CC1E XFERAD  EQU   $CC1E
CC20 ERRTYP  EQU   $CC20
CC21 SPECIO  EQU   $CC21
CC22 OUTSW   EQU   $CC22
CC23 INSW    EQU   $CC23
CC24 OUTFIL  EQU   $CC24
CC26 INFIL   EQU   $CC26
CC28 CMDFLG  EQU   $CC28
CC29 CURCLN  EQU   $CC29
CC2B MEMEND  EQU   $CC2B
CC2D ERRVEC  EQU   $CC2D
CC2F FILEKO  EQU   $CC2F
CC33 CPUTYP  EQU   $CC33
CC35 PRTADR  EQU   $CC35
CC37 PRTLNG  EQU   $CC37
CC39 PRTDVC  EQU   $CC39
CCC0 PINIT   EQU   $CCC0
CCD8 PCHK    EQU   $CCD8
CCE4 POUT    EQU   $CCE4

                     *FLEX USER-CALLABLE ROUTINES

                     CD00 COLDS   EQU   $CD00
                     CD03 WARMS   EQU   $CD03
                     CD06 RENTER  EQU   $CD06
                     CD09 INCH    EQU   $CD09
                     CD0C INCH2   EQU   $CD0C
                     CD0F OUTCH   EQU   $CD0F
                     CD12 OUTCH2  EQU   $CD12
                     CD15 GETCHR  EQU   $CD15
                     CD18 PUTCHR  EQU   $CD18
                     CD1B INBUFF  EQU   $CD1B
                     CD1E PSTRNG  EQU   $CD1E
                     CD21 CLASS   EQU   $CD21
                     CD24 PCRLF   EQU   $CD24
                     CD27 NXTCH   EQU   $CD27
                     CD2A PSTRIO  EQU   $CD2A
                     CD2D GETFIL  EQU   $CD2D
                     CD30 LOAD    EQU   $CD30
                     CD33 SETEXT  EQU   $CD33
                     CD36 ADDBT   EQU   $CD36
                     CD39 OUTDEC  EQU   $CD39
                     CD3C OUTHEX  EQU   $CD3C
                     CD3F RPTERR  EQU   $CD3F
                     CD42 GETHEX  EQU   $CD42
                     CD45 OUTADR  EQU   $CD45
                     CD48 INDEC   EQU   $CD48
                     CD4B DOCMND  EQU   $CD4B
                     CD4E STAT    EQU   $CD4E

                     *FMS ADDRESSES

C840 SYSFCB  EQU   $C840
D400 FMSINI  EQU   $D400
D403 FMSCLS  EQU   $D403
D406 FMS     EQU   $D406
D409 BASFCB  EQU   $D409
D40B CURFCB  EQU   $D40B
D435 VRFYFG  EQU   $D435

                     END

                     *THIS PROGRAM WILL ALLOW YOU TO LOCK UP
                     *THE COMPUTER UNTIL THE CORRECT CODE
                     *(WHICH YOU INSERT AT ASSEMBLY TIME)
                     *IS ENTERED IN THE ALLOTED TIME.
```

```
                        ********************************
                        * Written By  MR. ED. DeLAUTER *
                        ******** June 1983 **************

                        * SYNTAX: LOCK

C100                            ORG   $C100

C100 20  63             START   BRA   START

C102 01                         FCB   1         Version 1

C103 38 33 37 32        CODE    FCC   '83726871' Your code sequence 21 characters
C107 36 38 37
C10A 0D                         FCB   $0D


C10B 10                 CLRSCR  FCB   $10       HOME UP CURSOR
C10C 16                         FCB   $16       ERASE TO END OF PAGE
C10D 1E 15                      FCB   $1E,$15   SUPPRESS CURSOR DISPLAY
C10F 04                         FCB   $04
C110 1C 12              CLRSCR  FCB   $1C,$12   SET CRT FORMAT II
C112 1E 14                      FCB   $1E,$14   SET UNDERLINE CURSOR
C114 1E 03                      FCB   $1E,$03   SET BLINKING CURSOR
C116 04                         FCB   $04

C117 49 4E 50 55        TXT1    FCC   'INPUT YOUR USER CODE'
C11B 54 20 59 4F
C11F 55 52 20 55
C123 53 45 52 20
C127 43 4F 44 45
C12B 04                         FCB   $04
C12C 54 48 45 20        TXT2    FCC   'THE CODE YOU ENTERED IS INVALID'
C130 43 4F 44 45
C134 20 59 4F 55
C138 20 45 4E 54
C13C 45 52 45 44
C140 20 49 53 20
C144 49 4E 56 41
C148 4C 49 44
C148 04                         FCB   $04


C14C                    ICODE   RMB   24
C164                    IOFTMP  RMB   1         TEMP STORAGE FOR I/O FLAG

                        *PROGRAM STARTS HERE

C165 8E  C10B           START   LDX   #BLKSCR   Blank screen
C168 BD  6A                     BSR   OUTCOD
C16A 9D  CD4E           START3  JSR   STAT      Wait for Key Stroke
C16D 27  06                     BEQ   START4    then proceed...
C16F AD  9F D3E5                JSR   [$D3E5]
C173 20  F5                     BRA   START3
C175 AD  9F D3E5        START4  JSR   [$D3E5]   Throw away input Character
C179 8E  C110                   LDX   #CLRSCR   Clear (restore) Terminal
C17C BD  56                     BSR   OUTCOD
C17E 8E  C117           START2  LDX   #TXT1     Enter Code message
C181 BD  51                     BSR   OUTCOD    Output message, CrLf, CrLf
C183 BD  CD24                   JSR   PCRLF
C186 BD  CD24                   JSR   PCRLF
C189 20  06                     BRA   IPC       Goto Input Code Subroutine
C18B BD  CD24           START1  JSR   PCRLF     Print CrLf & Return to FLEX
C18E 7E  C003                   JMP   WARMS

                        *INPUT PASS CODE

C191 108E C14C          IPC     LDY   #ICODE
C195 C6  08             IPC2    LDB   #8        Loop count 2
C197 8E  6000           IPC4    LDX   #$6000    Loop count 1
C19A BD  CD4E           IPC1    JSR   STAT      Wait for Key Stroke
C19D 26  0B                     BNE   IPC3
C19F 30  1F                     LEAX  -1,X      Decrement Loop count 1
C1A1 26  F7                     BNE   IPC1
C1A3 BD  54                     BSR   BELL      Ring bell if loop 1 counts to zero
C1A5 5A                         DECB            Decrement Loop count 2
C1A6 26  EF                     BNE   IPC4
C1A8 20  BB                     BRA   START     Re-start if loop 2 counts to zero
C1AA AD  9F D3E5        IPC3    JSR   [$D3E5]   Get input character
C1AE A7  A0                     STA   ,Y+       Store in ICODE buffer
C1B0 108C C161                  CMPY  #ICODE+21 Check for buffer overflow
C1B4 27  2E                     BEQ   ERROR
C1B6 81  0D                     CMPA  #$0D
C1B8 27  07                     BEQ   IPC5
C1BA 86  58                     LDA   #'X       Output ASCII 'X' to terminal
C1BC BD  CD0F                   JSR   OUTCH
C1BF 20  04                     BRA   IPC2
C1C1 8E  C103           IPC5    LDX   #CODE     Set-up pointers.
C1C4 108E C14C                  LDY   #ICODE
C1C8 A6  80             IPC6    LDA   ,X+       Compare codes, Check for equal
C1CA A1  A0                     CMPA  ,Y+
C1CC 26  16                     BNE   ERROR     If they don't match goto ERROR
C1CE 81  0D                     CMPA  #$0D
C1D0 27  B9                     BEQ   START1
C1D2 20  F4                     BRA   IPC6

                        *OUTPUT CODE FOR SETUP

C1D4 A6  80             OUTCOD  LDA   ,X+       Get character for output
C1D6 81  04                     CMPA  #$04
C1D8 27  09                     BEQ   OCS1      Branch if EOL (04)
C1DA 81  0D                     CMPA  #$0D
C1DC 27  05                     BEQ   OCS1      Branch if CR (0D)
C1DE BD  CD0F                   JSR   OUTCH     Output char. in A reg.
C1E1 20  F1                     BRA   OUTCOD    Continue till done
C1E3 39                 OCS1    RTS

                        *ERROR HANDLING ROUTINE

C1E4 BD  CD24           ERROR   JSR   PCRLF     Wrong Code has been input
C1E7 BD  CD24                   JSR   PCRLF     print error message, Ring bell
C1EA 8E  C12C                   LDX   #TXT2     and branch to enter code again...
C1ED BD  E5                     BSR   OUTCOD
C1EF BD  CD24                   JSR   PCRLF
C1F2 BD  CD24                   JSR   PCRLF
C1F5 8D  02                     BSR   BELL
C1F7 20  85                     BRA   START2

                        *RING BELL

C1F9 B6  07             BELL    LDA   #$07     Ring Bell
C1FB 7E  CD0F                   JMP   OUTCH

                                END   START

0 ERROR(S) DETECTED
```

PL/9 - A BASIC COMPARED

Comparison of Windrush PL/9 and Microware A/BASIC
for Dedicated Software Development on the MC6809

by E. M. (Bud) Pass, Ph.D.
Computer Systems Consultants, Inc.
1454 Letta Lane, Conyers GA 30207
Telephone Number 404-483-1717/4570

INTRODUCTION

PL/9 and A/BASIC are programming systems designed primarily for dedicated, special purpose, applications which would normally be developed using assembler language.

The PL/9 system is composed of the following three parts, which are coresident in memory during program development.

- a line editor,
- a compiler,
- a symbolic debugger.

The A/BASIC system is composed of only a compiler.

This article compares PL/9 and A/BASIC in terms of their use in such applications as dedicated development in which speed and size are more important factors than language compatibility and ease of development, although these factors are always desirable. The discussion will concentrate on the PL/9 compiler language itself, mentioning the PL/9 editor and debugger only briefly.

LANGUAGE COMPARISONS

GENERAL

The PL/9 language is a not-very-high-level language, more similar to "C" than it is to BASIC. WINDRUSH claims that PL/9 is a high-level language, similar to PL/M or PASCAL; however, it has no string nor input/output constructs and cannot therefore be logically classified as a high-level language. It supports

GOTO Page #42

ARITHMETIC EXPRESSIONS

PL/9 evaluates arithmetic expressions in a left to right fashion, without regard to operator precedence, unlike most other languages. This trait may cause major problems for those already familiar with BASIC. However, PL/9 honors parentheses to force other orders of evaluation. PL/9 supports the following operators in top to bottom (but not left to right) hierarchial order:

1. (,)
2. functions, unary -, pointer .
3. *, /, +, -, AND, EOR, OR, =, <>, >, <, >=, <=, =>, =<

PL/9 provides the pre-defined constants TRUE and FALSE and variables ACCA, ACCB, ACCD, CCR, and MEM for the convenience of the user. TRUE has value of -1 and FALSE has value of 0. ACCA, ACCB, ACCD, and CCR represent the 6809 A, B, D, and CC registers, respectively. MEM is a BYTE array based at zero, with size 65536, so all of memory may be addressed directly as if it were an array.

A/BASIC evaluates arithmetic expressions with regard to operator precedence, as is common with most other languages. It supports the following operators in top to bottom (but not left to right) hierarchial order:

1. (,)
2. functions
3. @, unary +
4. &, |, ^
5. *, /
6. +, -, string +
7. =, <>, >, <, >=, <=, =>, =<

INPUT/OUTPUT

A/BASIC has syntactical elements defining access to the terminal and disk files, along with the scanners and number converters required to get data in and out of the program to those devices. PL/9 has no syntactical elements defining input/output to any devices, nor does it have the support functions to support the number conversion. Thus, A/BASIC input/output will be substantially easier to use for those applications involving terminal and disk files. However, for other applications, both languages have constructs which support direct access to memory and the calling of external and internal machine language routines to assist in the support of other devices.

Both A/BASIC and PL/9 support the GEN statement, which allows the direct insertion of machine language into the program. PL/9 requires that GEN statements be grouped into ASMPROCs, which are otherwise similar in declaration to PROCEDUREs. In an ASMPROC, it is not necessary to name the parameters, the declaration of local variables is not allowed, and the type of the ASMPROC, if any, must be explicitly stated following a colon following the parameter list, if any.

A/BASIC allows GEN statements at any point which any other statement may appear. PL/9 GEN statements generate one byte per entry, whereas A/BASIC GEN statements generate two bytes per entry if the constant value is greater than 255 and one byte otherwise. The PL/9 companion assembler, MACE, separately available from Windrush, has an option which allows it to generate the ASMPROC declarations and GEN statements from assembler language input.

PL/9 supplies the MEM array automatically. It is a BYTE array of 65535 elements, representing all of memory. Thus any location in the address space may be inspected and modified. A similar array could be declared in A/BASIC by the user, or the A/BASIC PEEK function or POKE statement could be used for similar purposes.

Both languages also support the declaration of variables at named specific addresses in memory, and this could be used to logically inspect and modify those locations. For example, a PIA at $E030 could be pointed to in an A/BASIC program as follows:

```
BASE=$E011
DIM P$(4)
```

and the same PIA could be pointed to in a PL/9 program as follows:

```
AT $E030: BYTE PIA(4)
```

Then P$(n) and PIA(n) would access the n-th internal PIA register, where n has a value of 0, 1, 2, or 3.

Neither language has an absolute advantage over the other in terms of supporting special devices, although A/BASIC has a decided advantage over PL/9 in terms of support for terminal and disk access, at least when generating code to run under control of an operating system such as FLEX or OS/9.

DEBUGGING AIDS

The A/BASIC compiler provides the normal careful checking of the syntax of an A/BASIC program provided by most other BASIC compilers. However, it does not provide the careful run-time checking of the execution of an A/BASIC program which users of many BASIC interpreters have come to expect, nor does it provide such debugging aids as line or variable traces or the setting of breakpoints at specified lines. The usual method of modifying a program to assist in debugging it is to add PRINT statements to indicate the values of critical variables at critical points.

The PL/9 system consists of the PL/9 compiler, a line editor, and a symbolic debugger, all of which are co-resident in memory simultaneously. The generated object program may be placed into memory by the compiler for immediate debugging or execution or on disk for later execution. The editor may be used to create or modify PL/9 or other programs and data files. PL/9 programs may be checked for syntax and certain run-time errors and easily corrected, without leaving the PL/9 system. The compiler saves the names and locations of the variables and inserts calls to the symbolic debugger in the PL/9 program when it is invoked with the symbolic debugger option. Then, when the program is executed under control of the symbolic debugger, it can be single-stepped, breakpointed, traced, and its variables may be printed by name. The debugging code is not inserted into the PL/9 library portion of the program, only into the user portion of the program. It cannot catch all classes of errors, and some, such as invalid subscripts, may crash the entire system; nevertheless, it is very useful as a debugging tool.

SUMMARY

This article has compared the Windrush PL/9 and Microware A/BASIC systems in terms of development of dedicated systems. The PL/9 system has the advantages of supporting a structured language and having a co-resident editor and symbolic debugger. The A/BASIC system has the advantage of supporting terminal and disk files and string manipulation within the syntax of the language.

PL/9 is available from Windrush Micro Systems for $198.00, and MACE is available for $98.00. A/BASIC is available from Frank Hogg Laboratory for $150.00.

APPENDIX 1: SYNTAX AND SEMANTICS CHARTS

The following tables present the syntax and semantics for each of the languages A/BASIC and PL/9. The tables are divided into functions, operators, and statements. The following notation is used in the tables:

b	byte expression
B	byte constant
e	numeric or string expression
L	list of variables and string variables
n	numeric expression
N	numeric constant
s	statement
s$	string expression
S$	string constant
v	variable
v$	string variable
x	numeric or Boolean expression
[ ]	optional
,...	repeated

The first column classifies the syntactic token on the line. The second column contains the name of the function, operator, or statement. The third column contains the parameters of the function, operator, or statement. The next one or more lines contain the semantics associated with each function, operator, or statement.

A/BASIC SYNTAX AND SEMANTICS

TYPE	NAME	PARAMETERS SEMANTICS
function	ABS	(n) absolute value of n
function	ASC	(a$) numeric value of first character of a$
function	BUF$	 I/O buffer
function	CHR$	(n) ASCII character corresponding to n
function	EOF	(@n) test file n for end of file condition
function	ERR	 error number
function	FILSIZ	(@n) number of sectors in file n
function	LEFT$	(a$,n) string representing n characters starting at left of a$
function	LEN	(a$) length in bytes of a$
function	MID$	(a$,n[,n]) string representing n2 (or remaining) characters starting at n1 characters into a$
function	PEEK	(n) 8-bit numeric value at address n
function	POS	 character position in print buffer
function	RIGHT$	(a$,n) string representing n characters at at the end of a$
function	RND	[(n)] random number between 0 and 32767
function	STATUS	(@n) status of file n
function	STR$	(n) string conversion of numeric n
function	SUBSTR	(a$,a$) first occurrence of a1$ in a2$ (or zero if not found)
function	SWAP	(n) swapped bytes of 16-bit value n

```
function  TAB      (n)
                   advance print buffer pointer to
                   position n
function  TRM$     (a$)
                   argument a$ without trailing spaces
function  VAL      (a$)
                   numeric conversion of string a$
operator  |
                   binary logical inclusive or (l|r)
operator  "        s$"
                   unary string constant definition
operator  !
                   unary logical negation (!r)
operator  $
                   unary hexadecimal constant definition
operator  %
                   binary logical exclusive or (l%r)
operator  &
                   binary logical and (l&r)
operator  (
                   unary expression group start
operator  (#
                   unary file number group start
operator  )
                   unary expression group end
operator  *
                   binary numeric multiply (l*r)
operator  +
                   binary numeric addition (l+r),
                   binary string concatenation (l$+r$),
                   unary numeric positive (+r)
operator  ,
                   binary subscript separator,
                   binary parameter separator,
                   binary PRINT punctuation (tab),
                   unary PRINT punctuation (CRLF)
operator  -
                   binary numeric subtraction (l-r),
                   unary numeric negation (-r)
operator  /
                   binary numeric division (l/r)
operator  :
                   binary separate statements on line
operator  ;
                   binary PRINT punctuation (no tab),
                   unary PRINT punctuation (no CRLF)
operator  <
                   binary numeric less (l<r),
                   binary string less (l$<r$)
operator  <=
                   binary numeric not greater (l<=r),
                   binary string not greater (l$<=r$)
operator  <>
                   binary numeric not equal (l<>r),
                   binary string not equal (l$<>r$)
operator  =
                   binary numeric equal (l=r),
                   binary string equal (l$=r$)
                   binary numeric assignment (l=r),
                   binary string assignment (l$=r$)
operator  =<
                   binary numeric not greater (l=<r),
                   binary string not greater (l$=<r$)
operator  =>
                   binary numeric not less (l=>r),
                   binary string not less (l$=>r$)
operator  >
                   binary numeric greater (l>r),
                   binary string greater (l$>r$)
operator  >=
                   binary numeric not less (l>=r),
                   binary string not less (l$>=r$)
statement *        ...
                   introduce remark (column 1)
statement .        ...
                   introduce remark (column 1)
statement BASE     [=]n
                   set ram assignment address to n.
statement CALL     n
                   call machine language subroutine
                   at address n
statement CHAIN    a$
                   branch to label v
statement IF       x THEN S [ELSE S]
                   perform statement S1 if expression x
                   true; otherwise perform statement S2
statement INCLUDE  S$
                   compile file S$ as part of PL/9 program
statement JUMP     N
                   branch to machine language subroutine
                   at address N
statement ORIGIN   =N
                   set program assignment address to N
statement PROCEDURE ...
                   introduce PL/9 subroutine
statement v[(n)]   =n
                   assign expression on right of equal
                   to variable on left
statement REPEAT   S FOREVER
                   repeat statement S endlessly
statement REPEAT   S UNTIL x
                   repeat statement S until x is true
statement RETURN   [n]
                   terminate execution of PROCEDURE
                   and optionally returns value and
                   type represented by n
statement STACK    =N
                   set stack assignment address to N
statement STACK    =*
                   set stack assignment address to
                   load and run BASIC program named a$
statement CLOSE    #n[,#n[,...]]
                   close specified files n1, n2, ...
statement CLOSE    FILES
                   close all files
statement CREATE   #n,a$
                   create file n with name a$
statement DIM      v(n[,n])[,...]
                   declare dimensioned variables
statement DIM      v$(n[,n])[,...]
                   declare string variables with length n1
                   or string arrays of dimension n1,
                   length n2
statement DISPLAY  [#n[,]][a[;]][,...]
                   output strings and control characters
                   to terminal
statement END
                   terminate execution
statement EXPAND
                   expand memory space
statement FOR      ven TO n [STEP n]
                   create loop with control variable v
                   set initially to n1, terminal condition
                   of v crossing n2, step size n3 (or 1)
statement GEN      N[,N[,...]]
                   insert specified values into program
statement GOSUB    N
                   call subroutine starting at line N
statement GOTO     N
                   branch to line N
statement IF       x GOSUB N
                   call subroutine starting at line N
                   if expression x true
statement IF       x THEN N
                   branch to line N if expression x true
statement IF       x THEN S [ELSE S]
                   perform statement S1 if expression x
                   true; otherwise perform statement S2
statement INPUT    L
                   input list L from terminal
statement KILL     a$
                   delete file named a$
statement [LET]    BUF$=a$
                   replace contents of I/O buffer with
                   string expression to right of equal
statement [LET]    v$[(n)]=a$
                   assign expression on right of equal
                   to variable on left
statement [LET]    v[(n[,n])]=n
                   assign expression on right of equal
                   to variable on left
statement NAME     S$
                   set module name to S$
statement NEXT     v
                   initiate next iteration for FOR loop
                   with control variable v
statement ON       ERROR GOTO [N]
                   set trap at line N for error handling
                   or terminate error handling trap
statement ON       NOVR GOTO N
                   branch to line N if no overflow
statement ON       OVR GOTO N
                   branch to line N if overflow
statement ON       n GOSUB N[,N[,...]]
                   call subroutine at n-th line number N
statement ON       n GOTO N[,N[,...]]
                   branch to n-th line number N
statement OPEN     #n,a$
                   open file n1 with name a$
statement OPT      L
                   provide compile options
statement ORG      [=]n
                   set program address to n
statement PAG
                   continue compiler listing on next page
statement POKE     (n)=n
                   store 8-bit value n2 at address n1
statement PRINT    [#n[,]][a[;]][,...]
                   output characters to terminal
statement READ     #n,L
                   read data into list L from
                   sequential file n
statement REM
                   introduce remark
statement RESTORE  #n[,#n[,...]]
                   rewind files n1, n2, ... and
                   reopen for input
statement RETURN
                   return from most recent active GOSUB
statement RREAD    #n,n,L
                   read data into list L from
                   random file n1 record n2
statement RWRITE   #n,n,L
                   write data from list L into
                   random file n1 record n2
statement SCRATCH  #n[,#n[,...]]
                   rewind files n1, n2, ... and
                   reopen for output
statement SHELL    a$
                   pass message a$ to operating system
statement STACK    [=]n
                   set initial stack pointer to address n
statement STOP
                   terminate program execution
statement WRITE    #n,L
                   write data from list L into
                   sequential file n
```

PL/9 SYNTAX AND SEMANTICS

```
TYPE        NAME      PARAMETERS   SEMANTICS

function    ACCA
                                   contents of A register
function    ACCB
                                   contents of B register
function    ACCD
                                   contents of D register
function    BYTE      (n)
                                   byte equivalent of integer n
function    CCR
                                   contents of CC register
function    EXTEND    (n)
                                   16b of n sign-extended to first byte
function    INTEGER   (b)
                                   integer equivalent of byte b
function    MEM       (n)
                                   contents of address n
function    NOT       (n)
                                   ones-complement of n
function    SWAP      (n)
                                   swapped bytes of 16-bit value n
operator    '         B
                                   unary byte constant definition
operator    (
                                   unary expression group start
operator    )
                                   unary expression group end
operator    *
                                   binary multiply (l*r)
operator    +
                                   binary addition (l+r),
                                   unary positive (+r)
operator    -
                                   binary subtraction (l-r),
                                   unary negation (-r)
operator    .         v
                                   unary pointer to v
operator    /
                                   binary division (l/r)
operator    :
                                   binary separate declarations
operator    ;
                                   binary separate statements
operator    <
                                   binary less (l<r)
operator    <=
                                   binary not greater (l<=r)
operator    <>
                                   binary not equal (l<>r)
operator    =
                                   binary equal (l=r)
operator    >
                                   binary greater (l>r)
operator    >=
                                   binary not less (l>=r)
statement   /*        ... */
                                   introduce remark
statement   ACCA      =n
                                   set contents of A register to n
statement   ACCB      =n
                                   set contents of B register to n
statement   ACCD      =n
                                   set contents of D register to n
statement   ASMPROC
                                   introduces machine language PROCEDURE
statement   AT        N;
                                   set variable assignment address to N
statement   BEGIN     [S;[S;[,...]]] END
                                   group statements into one unit
statement   BREAK
                                   terminate WHILE or REPEAT statement
statement   CALL      N
                                   call machine language subroutine
                                   at address N
statement   CCR       =n
                                   set contents of CC register to n
statement   CONSTANT  v=N[,v=N[,...]]
                                   introduce list of constants
statement   END
                                   terminate statement group started
                                   with BEGIN
statement   ENDPROC   [n]
                                   end PROCEDURE and optionally returns
                                   value and type represented by n
statement   GEN       N[,N[,...]]
                                   insert specified values into program
statement   GLOBAL    ...
                                   introduce declaration of global variables
statement   GOTO       v
                                   current program assignment address
statement   WHILE     x S
                                   while x is true repeat statement S
```

APPENDIX 2: A/BASIC AND PL/9 EXAMPLES

Following are two examples representing (almost) the same algorithm programmed in each of the languages A/BASIC and PL/9.

A/BASIC EXAMPLE

```
0100 REM A/BASIC VERSION OF ERATOSTHENES SIEVE
0200 REM
0300 ORG=$A000
0400 STACK=$9FFF
0500 REM
0600 DIM F(255)
0700 DISPLAY $07;
0800 S=255
0900 FOR M=1 TO 10
1000 C=0
1100 FOR I=1 TO S:F(I)=1:NEXT I
1200 K=4
1300 L=0
1400 FOR I=1 TO S
1500 IF F(I)=0 THEN 2000
1600 P=I+I+1
1700 C=C+1
1800 PRINT P;" ";
1900 IF L=0 THEN FOR J=K TO S STEP P:F(J)=0:NE T J
2000 IF L=1 THEN 2300
2100 K=K+I+1+I+I+4
2200 IF K>S THEN L=1
2300 NEXT I
2400 NEXT M
2500 DISPLAY $07
2600 STOP
2700 END
```

PL/9 EXAMPLE

```
/* PL/9 VERSION OF ERATOSTHENES SIEVE */

ORIGIN=$A000;
STACK=*;

INCLUDE IOSUBS.LIB;

ASMPROC OUTNUM (INTEGER); /* OUTPUT NUMBER TO TERMINAL */
 GEN $AE,$62,$C6,$FF,$7E,$CD,$39;

PROCEDURE SIEVE;
  INTEGER SIZE,PRIME,COUNT,LOOPS,I,J,K;
  BYTE FLAGS(8192),LAST;
  PUTCHAR($07); /* BEEP FOR SIGNON */
  SIZE=8191;
  LOOPS=1;
  WHILE LOOPS<=10 BEGIN /* LOOP 10 TIMES */
    LOOPS=LOOPS+1;
    COUNT=0;
    I=0;
    WHILE I<=SIZE BEGIN FLAGS(I)=TRUE; I=I+1; END;
    K=4;
    LAST=FALSE;

    I=1;
    WHILE I<=SIZE BEGIN
      IF FLAGS(I) THEN BEGIN
        PRIME=I+I+1;
        COUNT=COUNT+1;
        OUTNUM(.PRIME);
        IF NOT(LAST) THEN BEGIN
          J=K;
          WHILE J<=SIZE BEGIN FLAGS(J)=FALSE; J=J+PRIME; END;
        END;
      END;
      IF NOT(LAST)THEN BEGIN
        K=K+I+1+I+I+4;
        IF K>SIZE THEN LAST=TRUE;
      END;
      I=I+1;
    END;
  END;
  PUTCHAR($07); /* BEEP FOR SIGNOFF */
ENDPROC;
```

BIT BUCKET

MICROWARE. PRESS RELEASE

FOR IMMEDIATE RELEASE
Further information contact:
K. Jeanne Kaplan
515-279-8844

MICROWARE BREAKS GROUND FOR NEW FACILITY

Saturday October 9, 1981 is a date that will go down in 6809 microprocessor history: The staff of Microware held an informal ground breaking ceremony to mark the beginning of a new era for the company.

A 7500 square foot building, designed with special software development labs, demonstration areas and business offices is now under construction in Des Moines, Iowa, with a planned occupancy of March, 1984.

The first shovel of dirt was dug by Ken Kaplan, President of Microware, to usher in the "coming of age" of the Company. The programming staff took the honors of the second shovel followed by the marketing staff. Invited guests included members of the banking and real estate communities of Des Moines who assisted in the planning and financing.

"This marks a landmark in the history of Microware. The growth of the company has been on the UP and UP over the last two years and this building is the proof that we intend to keep up the momentum of bigger and better things." Ken Kaplan stated as the champagne glasses were lifted.

WESTCHESTER Applied Business Systems
Post Office Box 187
Briarcliff Manor, N.Y. 10510

PRODUCT ANNOUNCEMENT
Effective October 1, 1983

XDMS V1.1 and XDMS+

XDMS V1.1 is the first major revision to the XDMS Data Management System. This version incorporates several enhancements designed to add processing flexibility within the GENERATE process control statements. For example, it is now possible to do rate table lookups (eg; tax, FICA, etc) on ranges of values and incorporate the found values in subsequent calculations. Immediate commands have been added to permit in-process file deletes or purge, terminal messages and prompts, and optional escape from the process. GENERATE error handling is now non-fatal, allowing interactive users to correct or reenter instructions without reloading the program. XDMS V1.1 replaces V1.0 effective October 1, and is available for $179.95. Upgrades for registered XDMS V1.0 users is $25.00.

XDMS+ adds to V1.1 a set of useful utilities. SET permits display or change of the XDMS margin, width, page, ejects, backspace, end-of-line, delete, return echo, escape, reprint, null count, pause, backup, work drive and date values. PLOT produces horizontal bar, symbol or time charts. COPYDEF defines new files with existing formats. REVISE permits modification of file field definitions. OUTPUT dumps a .DMS file in tabular ASCII format. INPUT loads a .DMS file from dump output or user generated text. FORM displays the print form of a .DMS file. SIZE lists the number of file records and sectors. PURGE erases the data portion of a .DMS file. FILES displays a file name matrix for a given file extent. More to be added.

XDMS+ is available for $249.95. Registered XDMS V1.0 users may upgrade to XDMS+ for $50.00. Shipping on all orders is $2.50 domestic, $7.50 foreign. Orders may be placed with Southeast Media on 800-338-6800.

Universal Data Systems
MOTOROLA INC.

NEWS

MEDIA INFORMATION:
WILLIAM SCHLORER
Marketing Manager

Huntsville, Alabama, UNIVERSAL DATA SYSTEMS, (Huntsville, AL), a subsidiary of Motorola, Inc. (Information Systems Group), now offers high-speed, over-the-phone-line computer communications at a very competitive price. The Universal Data Systems model UDS 212LP 1200 Baud modem is available for $445 (suggested U.S. resale) at participating Universal Data Systems dealers.

The high speed of the UDS 212LP means less demand on computer time, lower line charges and other economies. This applies to use in communications between computers, dial-up information services, time-sharing systems, computer bulletin boards and so on.

The UDS 212LP has taken advantage of advances in engineering technology to greatly reduce its power requirements. Since it is powered entirely by the telephone line itself, the need for bulky, heat-producing power supplies or modules is eliminated. The result is a sleek, low-profile unit that rests comfortably under a telephone.

Universal Data Systems is located at 5000 Bradford Drive, Huntsville, Alabama, 35805. Telephone (205)837-8100.

TECHNICAL SPECIFICATIONS: UDS 212LP

The UDS 212LP offers 1200 bps differentially coherent phase shift keyed (PSK) communications under Bell 212 protocol. It is intended for use over two-wire direct-distance dial (DDD) public switched telephone networks.

Controls and indicators: A talk/data switch controls whether a telephone or modem is connected to the telephone line and an originate/answer switch, which selects the appropriate data exchange operating mode. A front panel light indicates when the modem is off-hook.

Dimensions: 6-1/8 inc. W by 9-1/2 in. D by 1-1/8 in. H.

Weight: 15 oz.

UDS modem functions are available in circuit-card form for OEM applications; UDS produces more integral modem cards for the data communications market than any other manufacturer — more than a half million modems during the past ten years.

Modems manufactured by Universal Data Systems are warranted to perform as specified for an entire year, assuming proper application and no physical or electrical abuse. During that period, UDS will repair or replace any modem that suffers a malfunction.

press release

Contact: Don Sinclevicz
Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, North Carolina 27514

(919) 493-1451

UniFLEX™ BASIC 68000 PRECOMPILER FOR IMMEDIATE RELEASE

FOR THE UNIX™ OPERATING SYSTEMS

Chapel Hill - Technical Systems Consultants, Inc., has announced the availability of the UniFLEX™ BASIC 68000 Precompiler for the UNIX™ Operating Systems. Available for OEM licensing, the UniFLEX™ BASIC 68000 precompiler is designed to accept an expanded, improved syntax of BASIC source code and converts, or precompiles, the source to a standard syntax acceptable to the UniFLEX™ BASIC 68000 interpreter. Perhaps the most useful feature of the UniFLEX™ BASIC 68000 Precompiler is that it allows the use of variable names of unlimited length. These names can include letters, numbers, and the underscore character. For example, "year_to_date_sales" is a valid variable name. This feature alone permits the creation of BASIC source that is vastly more readable than standard BASIC.

With the precompiler, all BASIC line numbers are optional, and they can be replaced with line labels. Like variable names, these labels can be of unlimited length and can include letters, numbers, and the underscore character. The use of alphanumeric line labels makes "goto" and "gosub" statements much easier to read. For example, the precompiler statement, "gosub calculate_interest",
may be used instead of the standard BASIC statement "gosub 8460".

A simple form of string substitution, "string/macros", permits the definition of a string of text which can be given a name. When that name is later referenced in the source, it is replaced by the entire defined string. Several conditional compilation commands permit various portions of a source program to be conditionally precompiled. Other features include the following: a single BASIC statement, or logical line, can extend across multiple physical lines; comments can be imbedded within BASIC lines or can stand alone on separate lines; variable types can be defined so that suffixes need not be appended to each variable; and the user has complete control over the printed source listing that the precompiler optionally outputs.

The output of the UniFLEX™ BASIC 68000 Precompiler is a compressed program that the UniFLEX™ BASIC 68000 interpreter can run. The precompiled program cannot be listed or edited while in BASIC and, therefore, proprietary BASIC programs may be developed and distributed without the need to divulge the source code.

The UniFLEX™ BASIC Precompiler requires the UniFLEX™ BASIC interpreter package. Although not a true compiler, the UniFLEX™ BASIC Precompiler will add a new dimension to programming in BASIC resulting from a vast improvement in writing, understanding, and maintaining BASIC programs.

MP-C TO MP-T REVISITED

18 SEPTEMBER 1983
EO CALHOUN PO BOX 73426 FAIRBANKS, ALASKA
99707

AFTER READING OF THE MP-C TO MP-T MODIFICATION BY DONNIE WRIGHT ET AL (SEPT. MJ) I JUST COULDN'T WAIT TO TRY IT OUT. THIS MOD WAS JUST THE THING I NEEDED TO GET PRINTER SPOOLING WORKING AND WAS TAILOR MADE FOR MY SYSTEM. IN FACT, MR. WRIGHT MUST HAVE PEEKED INTO MY COMPUTER SHACK FOR BACKGROUND FOR HIS LETTER.

AS WITH MOST DEVELOPEMENTS, EACH SUCCEEDING PARTY MAKES THEIR OWN CONTRIBUTIONS. HERE ARE MINE.

FIRSTLY, THE MOD IS EVEN SIMPLER TO IMPLEMENT THAN WAS INDICATED. REFERRING TO THE INSTRUCTIONS AND DIAGRAM IN THE SEPTEMBER ISSUE (PGS. 26-27) STEPS 3 AND 4 ARE UNNECESSARY AS CONTROL OF THE IRQ GENERATOR CAN BE ACHIEVED VIA PB0 (PIN 10) WHICH IS ALREADY CONNECTED TO THE RESET INPUT OF THE DIVIDE COUNTER (IC3).

STEP 1 MAY ALSO BE UNNECESSARY SINCE P87 (PIN 17) IS TRI- STATED WHEN PROGRAMMED AS AN INPUT; HOWEVER, BY THE TIME I REALIZED THAT THE DEED HAD BEEN DONE. TOLD YOU I WAS IN A HURRY.

REGARDING STEP 7 (JUMPER CI TO CO) THAT IS EASILY DONE BY SHORTING (SOLDER BLOB WILL DO) IC3 (PIN 1) TO THE BAUD RATE SELECT JUMPER (D) WHICH IS ADJACENT TO IT.

SO, SHORTING PIA PINS 17 TO 18 AND SHORTING IC3 PIN 1 TO D THEN STRAPPING FOR IRQ AND 110 BAUD DOES THE TRICK!

THE CB2 LINE AND ASSOCIATED CIRCUITRY INTERESTED ME ALSO SO I PLUGGED AN LED BETWEEN READER CONTROL (JACK PIN 9) AND GROUND (PIN 1) SO AS TO HAVE A VISUAL INDICATION OF WHEN THE THING IS ON.

WE ARE NOW IN THE REALM OF SOFTWARE AND PLEASE NOTE THAT DEPENDING ON HOW YOU PROGRAM THE DEVICE THE LIGHT MAY BE ON WITH THE INTERRUPTS OFF OR VICE VERSA.

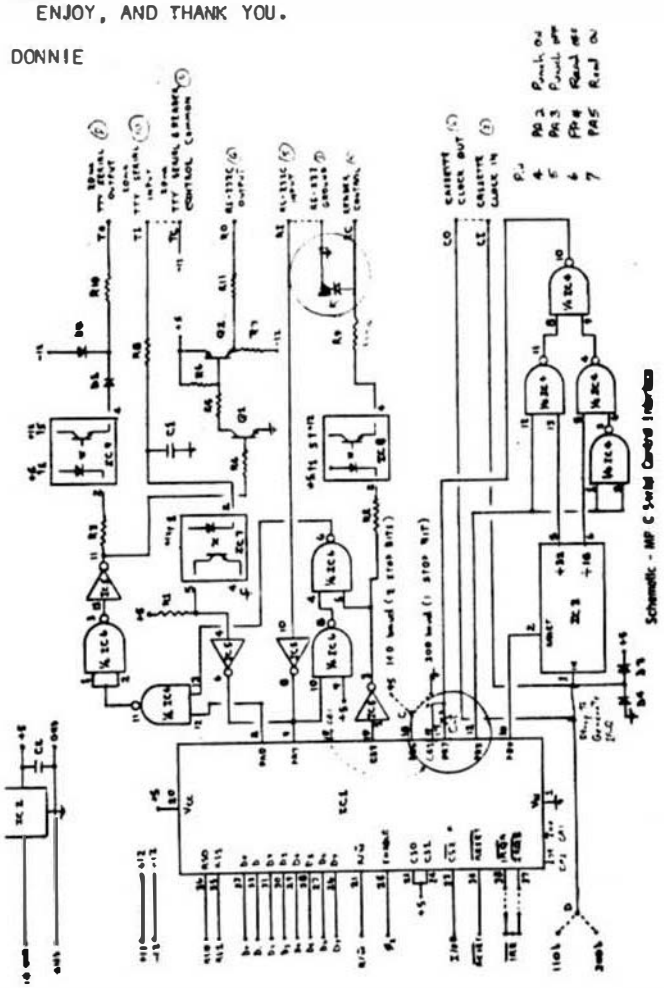
THE FOLLOWING LISTINGS SHOW THE PROGRAMMING TO HAVE THE LED AND INTERRUPTS ON OR OFF TOGETHER. THE SECOND LISTING IS A SIMPLE TESTING ROUTINE SO THAT YOU CAN CHECK THE FREQUENCY OF YOUR INTERRUPTS. DATA IN LOCATIONS $0000,$0001 WILL BE THE NUMBER OF IRQS' GENERATED PER INTERVAL (1 SEC IN LISTING).

BY THE WAY, YOU HAVE A CHOICE OF TWO IRQ INTERVALS AS DETERMINED BY THE STATE OF PB2. PB2=0 9 MSEC/IRQ OR PB2=1 18 MSEC/IRQ AT 110 BAUD. YOU CAN USE 300 BAUD TO GENERATE FASTER INTERRUPTS. PRETTY NEAT.

IN ACTUAL PRACTICE MY INTERRUPTS CAME AT THE RATE OF 108 PER SECOND (9.26 MS) AND 54 PER SECOND (18.6 MS).

ENJOY, AND THANK YOU.

DONNIE



A Special Thank You!

We want to express our thanks to all our customers. We have always felt that ***AAA Chicago Computer Center*** *has the best and most loyal customer base that any dealer has ever had. Many of our customers have grown with us since April 1977 when we first opened our doors in Chicago.*

Also, we would like to thank the manufacturers and vendors that we represent. Without their help and eager support, we never could have been able to supply our clients with the service that they have grown to expect from us. Together, we all form a team. Again, we want to acknowledge our thanks.

We'd like to note that our relationship with Smoke Signal Broadcasting and Southwest Technical Products dates all the way back to our beginning in April 1977. Not to slight anyone, we want to express thanks to our other vendors namely Technical Systems Consultants, Microware, Star-Kits, and our most recent addition, Helix.

At this time, we can say that we've probably installed more mixed SS-50 bus systems than any other dealer. For example, at the time we prepared this ad, we had 16 versions of FLEX (TM) drivers permitting the installation of our ELEKTRA Winchester systems with almost all versions of existing SS-50 and SS-30 floppy controllers.

While it is impossible and not economically feasible to support all combinations of hardware out in the field, we're trying our best. ***AAA Chicago Computer Center is the place to turn to when either adding on to your current SS-50 systems or purchasing a new one.***

New this month is our $125.00 auxiliary power supply permitting the installation of a second Winchester drive in our ELEKTRA computer cabinet. (Did you know that the ELEKTRA computer cabinet was the first SS-50 cabinet to include an EMI filter? The EMI filter has been included at no extra cost from the very beginning).

Did you know that our best selling 80 x 24 CRT terminal is the Hazeltine 1420? ***They are brand new and selling for only $375.00 plus shipping.***

Best wishes for a happy and healthy holiday season.

Phone
AAA Chicago Computer Center
Technical consultation available most weekdays from
4 p.m. to 6 p.m. C.S.T.
(312) 459-0450
120 Chestnut Lane, Wheeling, IL 60090

See our catalog and ordering information on the next page to your right.

HELIX

64K 6809 Computer	$2395.00	64K 68008 Computer	$2495.00
256K 6809 Computer	2895.00	256K 68008 Computer	2995.00

Other systems available

20 Megabyte 5" add on Winchester System			$2595.00
64K CMOS Static memory board with battery backup			395.00
DMA 5" and 8" Floppy Controller with built in Winchester controller I/O			695.00
DMA 5" and 8" Floppy Controller	$495.00	6809 CPU Board	495.00
68008 board for SS-50	595.00	CP/M-68K	350.00

Need FLEX, UniFLEX, OS-9 Level I, or OS-9 Level II? We have a system for you!

ELEKTRA COMPUTER CABINET THE LARGEST SS-50 COMPUTER CABINET AVAILABLE! Made of heavyweight 0.090" thick aluminum. Interior is 18-1/2" wide by 21-7/8" deep by 6-3/4" high. Heavy duty A.C. line cord. A.C. fuse holder. EMI filter. Fan with filter. Back panel has 10 cutouts for 'D' type data connectors. Front panel has key on/off power switch, 2 illuminated push button switches (Reset and NMI/Abort), and two cutouts for 5-1/4" disk drives. $250.00

Filler Plate for 5-1/4" drive opening: $10.00 Fan Filter: $10.00

POWER SUPPLY Highest quality linear power supply CONSERVATIVELY rated at 15a @ 8v, 3a @ 16v, 3a @ -16v. 3 primary inputs for light, rated, and heavy loading.
220v Version: $200.00 110v Version: $175.00

DISK REGULATOR BOARD WITH CABLES Standard version for 2 floppy drives $ 00
Heavy duty version for 1 Winchester drive and 1 floppy drive $75.00

AUXILIARY POWER SUPPLY to power second winchester drive $125.00

ENGINEER'S "FUN BOX" BY ELEKTRA Computer cabinet with high quality 10 amp power supply, EMI filter, fan. Large enough to hold the standard size SS-50 type motherboard. $225.00

ELEKTRA UNIVERSAL SS-50/SS-50C MOTHERBOARD Heavyweight 0.125" thick, 16" long by 9" wide, 11 memory (50 pin) slots, 8 I/O (30 pin) slots. Complete address decoding and selection, as well as extended address capability, for I/O slots. Choice of 4, 8, or 16 addresses per I/O slot. 1" spacing between all memory and I/O slots. On board baud rate generator with low and high ranges providing jumper selectable rates of 75 through 38,400 for each of the five baud rate lines. Slow device circuitry permitting 1 MHz 30 pin disk controllers to run with 2MHz 50 pin CPU boards.

Mounting hardware	$5.00	Bareboard w/documentation	$80.00
Kit w/gold connectors	$320.00	Assembled w/gold connectors	$380.00
Kit w/tin connectors	$240.00	Asembled w/tin connectors	$300.00

ELEKTRA CHASSIS Includes cabinet, 110v power supply, power supply cables, standard disk regulator board with power cables, motherboard with gold square pin connectors, assembled and tested. $850.00

ELEKTRA CPU 6/9 Use either the 6802 or 6809 (to run 6800 software) or 6809. Has provision for up to 3 2716 Eproms, 1K scratchpad, MC6840 triple timer, and an option 1 baud rate generator providing baud rates from 110 through 38,400 baud in two user selectable ranges. Versions of OS-9 level 1 are available.

Bareboard:	$50.00	Kit:	$225.00	Assembled:	$275.00
Optional Baud Rate Generator					$25.00

ELEKTRA DPS D ALPORT SERIAL CARD Fits the standard 30 pin SS-50 bus I/O slot. Can be configured for 4 or 16 addresses per port. RTS, CTS, OTR, DCD, IRQ, FIRQ/NMI, and baud rate can be appropriately implemented for each port.

Bareboard:	$20.00	Kit:	$60.00	Assembled:	$80.00
Cable with mounting hardware (two needed per board)				Each:	$25.00
Cable	Each: 20.00	Mounting hardware per cable			$5.00

ELEKTRA DPP DUAL PORT PARALLEL CARD Fits the standard 30 pin SS-50 bus I/O slot. Can be configured for 4 or 16 addresses per I/O slot. The direction of the TTL buffers can be controlled by either on board jumper connectors or by a signal from the peripherals. The interrupt request line for each port may be individually jumpered to either the IRQ or FIRQ/NMI bus line.

Bareboard:	$20.00	Kit:	60.00	Assembled:	$80.00
Cable with mounting hardware (two needed per board)				Each:	$25.00
Cable	Each: 20.00	Mounting hardware per cable			$5.00

ELEK RA 64K STATIC RAM/ROM MEMORY BOARDS with gold connectors (tin available) Assembled and tested. With 56K RAM $289.00 With 64K RAM $299.00

ELEKTRA UNIVERSAL SUPER FLOPPY CONTROLLER THE BEST 30 PIN FLOPPY DISK CONTROLLER THAT YOU CAN BUY! Controls up to four 5-1/4" drives and four 8" drives for a total of eight system drives. (FLEX system limit is four drives.) Single density or double density, 1MHz or 2MHz, 6800 or 6809. (Double density 8" must be at 2MHz; all other combinations of performance are possible.) Analog phase locked loop data separators with separate adjustments for 5" and 8" drives. Analog write precompensation circuit with separate adjustments for 5" and 8" drives. Designed to meet the data hold requirements of Western Digital floppy controller IC. Assembled and tested. $275.00

Disk with drivers and formatting utilities (Specify 6800/9, FLEX/OS-9) 30.00

ELEKTRA WINCHESTER SYSTEMS THE BEST WINCHESTER SYSTEMS THAT YOU CAN BUY! Has automatic error detection and CORRECTION of up to 11 bit burst errors. SS-50 bus, extended addressing capabilities, DMA, on board sector buffer, drivers included for 6809 FLEX or OS-9. Specify whose version of FLEX that you are using. Drivers for FLEX2 (6800) are available for an additional $100.00. Price includes host interface, controller, drive(s), and cables.

12 Megabyte single drive sys	$2295.00	24 Megabyte dual drive sys	$3595.00
19 Megabyte single drive sys	$2995.00	38 Megabyte dual drive sys	$4895.00

(19 Megabyte drives are the largest that can be supported by FLEX)

ELEKTRA HD-5 Cabinet for dual 5 1/4" floppy drives with power supply, line cord, fuse, power switch, and power cables to drives. 150.00

ELEKTRA HD-5W As above but with EMI filter, fan, and heavy duty power supply. 199.00

Powers 1 floppy and 1 Winchester.
5" ribbon cable for dual outboard 5-1/4" disk drives 40.00
2" ribbon cable for dual inboard 5-1/4" disk drives 35.00
Custom cables available Phone

ELEKTRA HD-8 Dual drive cabinet, EMI filter, fan, power supply, and power supply cables for 8" drives 350.00
8" ribbon cable for dual 8" disk drives 45.00

ELEKTRA 30 PIN PROTOTYPING BOARD 20.00
ELEKTRA 50 PIN PROTOTYPING BOARD 40.00

GOLD 10 PIN CONNECTORS (Specify male with square pins or female) 1.50
TIN 10 PIN CONNECTORS (Specify male with square pins or female) .50

ELEKTRA is a trademark of AAA Chicago Computer Center.
FLEX and **UniFLEX** are trademarks of Technical Systems Consultants, Inc.
HELIX is a trademark of Hazelwood Computer Systems.
OS-9 and **BASIC09** are trademarks of Microware Systems Corp.

Dealer for ELEKTRA, HELIX, SSB, SWTPC, Microware Systems Corp., and Technical Systems Consultants, Inc.

AAA CHICAGO COMPUTER CENTER
120 CHESTN T L NE • WHEELING IL 60090 (312) 459-0450

Technical consultation available 4 PM to 8 PM most weekdays. Closed evenings and weekends.

TERMS Minimum order $20.00. Shipping and handling estimates within the Continental U.S., add 3% (MINIMUM $2.50). Illinois residents add 6% sales tax. We will refund your overestimated shipping and handling charges. Foreign shipping and handling, add 10% (MINIMUM $10.00). Foreign orders must be prepaid in U.S. dollars. Checks must be drawn on a U.S. (or Canadian) bank. Heavy foreign items will be shipped air freight collect. Please phone between 4 PM and 6 PM weekdays if questions arise regarding shipping fees. Master Charge, Visa, and American Express honored.

Our apology: We are not staffed to answer technical inquiries through the mail. Please phone for technical help during the hours indicated above. The too frequent changing of our inventory and prices makes it uneconomical to publish a catalog. Our ads are intended to serve that purpose. Prices and inventory are subject to change without advance notice.

ELEKTRA™ SOFTWARE (All of our software is copyrighted and all rights are reserved. Source is either supplied or optionally available at extra cost so that the purchaser can modify our programs for his own use. Licensing, however, is required for commercial resale.)

SUPER MODEM PROGRAM Single character commands. No interrupts required. Transmit manually or transmit disk files (text) of any length to distant computer. Receive and save disk files (text) on local disk system. X-on/X-off supported. Tested for full duplex at speeds up to 9600 baud. Half duplex option. Echo option. Replaces CR with CR LF (user option). Slow disk file transmit option.

Please specify 6800 or 6809, SSB or FLEX™, 5" or 8".
Instruction Manual and disk with both source and object code $75.00

STANDARD MODEM PROGRAM Same as Super Modem Program above but without ECHO option, CR LF for CR option, slow disk file transmit option, nor X-on/X-off option. Specify 6800 or 6809.
Manual with instructions, source listing, and flow chart 10.00

OS-9 Configurable Modem Program (Sorry, source is not available) 100.00

ORDER — WRIT UP COMPUTER ROGRAM Screen oriented write up form with cursor editing, disk save and load, printer command using easily available universal print-out forms. Phone for more details. Available for 6809 FLEX. $100.00

ALL IN ONE
Editor — Text Processor — Mailing Labels — Mailing Lists — Multiple Form Letters
Use any CRT terminal and printer — Best Package For The Money Anywhere!
Specify 6800 or 6809, SSB or FLEX™, 5" or 8" 75.00
Printed source listing is available for an additional 35.00
All-in-One, Write'n Spell, and Spell'n Fix package 25.00

Software by Technical Systems Consultants, Inc.	**UniFLEX™ w/1 yr. mnt**	**FLEX™**
DOS (includes Editor and Assembler)	550.00	150.00
Editor or Assembler		50.00
68000 Cross Assembler on 6809	300.00	250.00
6800 FLEX™ Utilities		100.00
Text Processor or Sort/Merge Package or 6809 FLEX™ Utilities	150.00	75.00
Extended Basic	200.00	100.00
Basic Precompiler for Extended Basic	150.00	50.00
Pascal	300.00	200.00
Debug Package or Diagnostic Package	75.00	75.00
6809 Relocating Assembler & Linking Loader	175.00	150.00
Fortran (With Relocating Assembler & Linking Loader)	450.00	
Cobol	750.00	

Software by Microware By Items Corp.	**Run-Time Package**	**Update**	**Source**	**Manual Only**	**Object w/Man.**
OS-9™ Level One Operating System		75.00	600.00	40.00	200.00
OS-9™ Level Two Operating System		75.00	N/A	40.00	500.00
ASIC09™	100.00	75.00	N/A	25.00	200.00
OS-9™ Macro Text Editor			300.00	15.00	125.00
OS-9™ Interactive Assembler			300.00	10.00	125.00
OS-9™ Interactive Debugger (Disk version)			100.00	10.00	50.00
CIS Cobol Compiler	400.00	50.00	N/A	80.00	900.00
Pascal Compiler	100.00	100.00	N/A	40.00	400.00
"C" Compiler		100.00	N/A	40.00	400.00
Microware yearly support service ($200.00 for OS-9 Level 2)					75.00

Special Software			
2K 6809 MICROBUG	30.00	4K 6809 HUMBUG	75.00
2K 6800 HUMBUG	40.00	4K 6800 HUMBUG	65.00

Other HUMBUG versions including video versions are available.

Spell'n Fix by Peter Stark	178.58	Write'n Spell by Peter Stark	75.11
All-in-One, Spell'n Fix, and Write'n Spell package			250.00
Dynamite Disassembler			60.00
SUPER SLEUTH Disassembler System ($101.00 for OS-9 version)			99.00

SPECIALS

* U.S. Robotics 300/1200 baud auto dial/auto answer modem					499.00
* Same as above but without self test and diagnostics					399.00
* U.S. Robotics 1200 baud direct connect auto answer modem					349.00
* Hazeltine 1420 CRT terminal (new)					375.00
* SSB BFO Floppy Disk Controller (Version 3) Run FLEX or SSB DOS					100.00
* SWTPC 4K Memory	$15.00	MP-B	$40.00	MP-C	10.00
* MP-S	$60.00	MP-L	$60.00	MP-A	25.00
* High speed tape reader 50.00		AC-30	$40.00	MP-A2	125.00
* 300 Baud acoustic modem	129.00	F & D PMB-1 Video Assembled			60.00
* 10 ELEKTRA DD 8" Disks	35.00	10 5" DD Disks in hard box			25.00
* TI 810 Printer w/lower case and full vertical forms control					1200.00
* Microtime II Calendar and Clock Board (Assembled)					60.00

SD/DD DISK DRIVES	2 heads CDC	1 head MPI	2 heads MPI
30 day guarantee			
5-1/4", 40 tracks	300.00	250.00	325.00
5-1/4", 80 tracks	375.00	325.00	400.00
MPI or CDC Service Manual (Specify 40 or 80 track)			25.00
8", 77 tracks DS/DD	Qume DT-8 $550.00		Remex (Special) 350.00

GIMIX CLEARANCE SALE

	LIST	OUR PRICE		LIST	OUR PRICE
5 6809 Plus CPU Bd.	578.5	475.00	6800 CPU board	224.03	100.00
Cable (Ser or Par I/O)	24.9	20.00	Baud rate gen. board	88.93	65.00
Motherboard		200.00	8 Port Serial I/O Bd.	318.46	250.00
Double disk reg. card	68.22	50.00	426 control w/GMX Flex	328.28	270.00
32K memory board		175.00	56K memory board		425.00
64K memory board	478.67	450.00	Single prt ser. 1 cable	113.36	90.00
80 X 24 Video Boards	398.76	250.00	Dual prl par. 2 cables	136.32	110.00
64 X 16 Video Boards	198.71	100.00	4K PPD PROM Bd. and burner		100.00
16K Mem.Bds. w/cntrl reg.		145.00			

90L422 RAM chips (2 needed for GIMIX DAT) each 20.00

SWTPC	
6809 SWTPC FLEX™ and manual	35.00
DC-4 Disk Controller (SS/DS, SD/DD, 5-1/4")	230.00
MP-S2 Dual Port Serial	120.00
MP-L2 Dual Port Parallel	120.00
MP-N Calculator Board (kit)	54.95
MP-N (assembled)	92.00
MP-R 2716 Eprom Programmer	114.50
MP-09 2MHz 680 CPU Board	29.00

Smoke Signal Broadcasting	
DCB-4A Double Density Controller Board for 5" and 8" with DOS	549.00
SSB DOS (Specify 6800 or 6809, BFO or DCB-4A, 5" or 8")	75.00
SE92/SA92-5 (6809 Edit/Asem for DOS)	69.95
SSB Monitor (Specify 6800/6809, $8008/$E008/$F7E8)	75.00
SSB version of FLEX™ (without Editor and Assembler)	150.00
LMB-1A Motherboard	399.00
SCB-69 6809 CPU Board	399.00
PAR-1 Dual Port Parallel Board	89.00
SER-2 Dual Port Serial Board with 2 Cabl	129.00
Chief 90 64K Computer System	2195.00
Chief 95 4 64K Computer System with DS-DT-DD 5" FD	4325.00
Static Memory Boards M-16-X 195.00 M-24-X 295.00 M-32-X	395.00

WARNING AAA Chicago Computer Center does not provide repair or diagnostic service for customer assembled kits. AAA Chicago Computer Center does warranty and maintain service for our assembled boards. The customer should carefully take into consideration the small differential separating out kit and assembled prices when making his choice of purchase.

We have introduced our line of computer equipment with the purpose of offering the highest quality of components possible at affordable prices. These products are intended for OEM applications where it is the responsibility of the purchaser to integrate these components with suitable memory, disk controllers, drives, and software along with I/O terminals to form working computer systems.

```
* 18 SEPTEMBER 1983
*
* TESTMPC
*

*
* EQUATES
*

TINIT EQU $D397
TMON EQU $D3AD
TMOFF EQU $D39D
IRQVEC EQU $F3F8
MONITOR EQU $FC32 PSYMON

 ORG $A000

START LDX #IHNDLR
 STX IRQVEC
 CLRA
 TFR A,DP SET PAGE
 LDY #0
 STY 0 CLEARS IRQ COUNTER
 JSR TMOFF THIS DOES INIT WITHOUT
* ALTERING TEST VECTORS
 SEI JUST FOR BECAUSE
 JSR TMON START INTERRUPTS
 CLI READY, SET, GO
 LDX #1000 APPROX 1 SECOND
 BSR TIMER
 JSR TMOFF STOP IRQ
 JMP MONITOR
IHNDLR LDA $8002 CLEARS IRQ
 LDY 0 GET IRQ COUNTER
 LEAY 1,Y INCREMENT IT
 STY 0 SAVE IT
 RTI RETURN FROM IRQ
TIMER LDA #$5A A LOOP
CONT DECA EACH A LOOP APPROX.
 NOP EQUAL TO 1 MSEC
 NOP
 NOP
 BNE CONT
 DEX NEXT X
 BNE TIMER
 RTS

 END

* 17 SEPTEMBER 1983
*
* SPOOL PORT CONTROL ROUTINES
*
* ROUTINE CONTROLS SWTPC MP-C
* MODIFIED TO GENERATE INTERRUPTS
*
* TINIT INITIALIZES SYSTEM
* TMON TURNS ON IRQ GENERATOR
* TMOFF TURNS IT OFF
* IHNDLR CLEARS IRQ AND JUMPS
* TO FLEX CHANGE TASK ROUTINE
*
* IRQVEC AND SWIVEC ARE
* PSYMON EQUATES FOR
* PERCOM SBC/9 CPU

*
* EQUATES
*

PORT EQU $8000 PORT 0
FAST EQU $00 9 MSEC IRQ
SLOW EQU $04 18 MSEC IRQ
IRQVEC EQU $F3F8 *IRQ *
SWIVEC EQU $F3F2 *SWI3*
SWITCH EQU $C700 CHANGE TASK ROUTINE

 ORG $D397

TINIT LDX #IHNDLR
 STX IRQVEC
TMOFF LDX #PORT MP-C ADDRESS
 CLR 3,X RESET PIA
 LDA #$05 BITS 0&2=OUTPUT
 STA 2,X IN DDR
 STA 3,X SET PIA FOR IRQ
 LDA #$01 RESETS IRQ GENERATOR
 STA 2,X TURNS OFF INTERRUPT
 RTS
TMON LDX #PORT MP-C ADDRESS
 LDA #FAST BIT 2=0
 STA 2,X CONFIGURE IRQ
 LDA #$25 SET CB2 HIGH
 STA 3,X TURN ON LED
 LDA 2,X RESETS IRQ
 RTS
IHNDLR LDX #PORT MP-C ADDRESS
 LDA 2,X CLEARS IRQ
 JMP SWITCH

 END
```

BASIC SEARCH

JOHN A RAMSEY
8007 E. 166TH ST.
BELTON, MISSOURI 64012
PH 816-331-8350

13 AUG 1983

FOR MANY YEARS, I HAVE BEEN WRITING PROGRAMS IN BASIC USING MOST OF THE DIFFERENT VERSIONS PRODUCED BY SWTPC. IN ALL OF THE VERSIONS I HAVE USED, I HAVE NOTICED ONE SHORT-COMING THAT HAS HINDERED ME QUITE A BIT. THEY DON'T HAVE A SEARCH COMMAND.

HAVE YOU EVER WRITTEN A LONG PROGRAM AND HALF WAY THROUGH IT DISCOVERED YOU HAVE MISSPELLED A WORD THROUGHOUT THE ENTIRE PROGRAM? OR DECIDED, FOR ONE REASON OR ANOTHER, THAT YOU WANT TO CHANGE A VARIABLE FROM ONE CHARACTER TO ANOTHER? WELL I HAVE, AND TO FIND ALL THE USAGES OF THAT WORD OR VARIABLE, I HAVE TO SEARCH THE ENTIRE PROGRAM, LINE BY LINE, AND READ EACH WORD TO FIND ALL OF THE PLACES WHERE I HAVE USED THEM. THIS CAN BE QUITE TIME CONSUMING.

TO ALLEVIATE THIS PROBLEM, I HAVE WRITTEN A SEARCH ROUTINE FOR SWTPC BASIC VER. 3.0. I USE THIS BASIC WITH MY EXATRON STRINGY FLOPPY AND THEIR SIMPLEX-68 VER. 2.0 OPERATING SYSTEM FOR A 6800 MICROPROCESSOR. THIS SYSTEM, AS MOST EVERY ONE KNOWS, IS PRACTICALLY IDENTICAL TO THE TSC FLEX 2.0 DOS. REALIZING THAT MOST PEOPLE DON'T HAVE THE SIMPLEX-68 OPERATING SYSTEM, I HAVE CONVERTED THE ROUTINE ADDRESSES TO THOSE OF THE FLEX 2.0 SYSTEM.

SINCE MY VERSION OF BASIC WAS MODIFIED BY EXATRON FOR THE STRINGY FLOPPY, YOU MIGHT HAVE A PROBLEM LOCATING THE ADDRESSES FOR THE VRBBEG AND VRBEND EQU'S. I USE ONLY THE LAST HALF OF THE VERB TABLE IN MY ROUTINE AND THIS STARTS WITH THE VERB "LIST". THIS VERB IS PRECEDED IN MY BASIC WITH THE BYTES 00 99 99. START WITH THE ADDRESS OF THE FIRST BYTE AFTER 99 99 AS YOUR VRBBEG EQU. THE LAST VERB IN MY TABLE IS "STEP", FOLLOWED BY THE BYTES 00 12 90. THE ADDRESS OF THE BYTE 90 IS THE ADDRESS YOU WANT TO USE AS YOUR VRBEND EQU. AGAIN, BEAR IN MIND THAT THE BEGINING AND ENDING ADDRESSES OF THIS PART OF MY VERB TABLE MAY BE DIFFERENT FROM YOUR VERSION 3.0 SO ADJUST ACCORDINGLY. OTHER THAN THESE TWO ADDRESSES, YOU SHOULD HAVE NO PROBLEMS ADDING THIS ROUTINE TO YOUR BASIC.

LONG AGO, I RAN OUT OF ROOM IN THE VERB TABLE TO ADD NEW ROUTINES, SO I EXTENDED THE TABLE TO THE END OF BASIC. HOWEVER, SINCE MOST OF YOU PROBABLY HAVE NOT DONE THIS, I HAVE WRITTEN THIS PROGRAM TO REPLACE THE VERB "LET" WHICH WE DON'T NEED ANYWAY. AND ONCE AGAIN, IF YOUR "LET" VERB ADDRESS DOES NOT START AT THE SAME ADDRESS AS MINE, THEN SIMPLY FIND OUT WHERE IT DOES START AND CHANGE THE ADDRESS ACCORDINGLY.

TO USE THIS PROGRAM, TYPE "FIN" AND A CARRIAGE RETURN. YOU WILL GET A CRLF AND A QUESTION MARK PROMPT. YOU THEN TYPE IN THE CHARACTER, WORD OR WORDS YOU WANT TO FIND, FOLLOWED BY A CARRIAGE RETURN AND THE PROGRAM WILL LIST ALL LINES AND THEIR TEXT THAT CONTAIN THE INFORMATION YOU ARE SEEKING.

EXAMPLE:

```
FIN

? FREQUENCY

0020 PRINT TAB(20);"LOW FREQUENCY TRANSISTOR AMPLIFIER DESIGN"

0060 INPUT "CHANGE FREQUENCY (Y-N)";A$:IF A$="Y" THEN 330

BOTTOM
```

```
                    NAM   FIND
                *
                *   JOHN R RAMSEY                          13 AUG 1983
                *   8307 E. 166TH ST.
                *   BELTON, MISSOURI 64012
                *   PH 816-331-8350
                *****************************************************
                *                  EQUATES
                *****************************************************
A000            BUFFER  EQU   $A000     FLEX LINE BUFFER
AD1B            INBUFF  EQU   $AD1B
AD45            OUTADR  EQU   $AD45
AD18            PUTCHR  EQU   $AD18
AD24            PCRLF   EQU   $AD24
AD1E            PSTRNG  EQU   $AD1E
0082E           PRGBEG  EQU   $082E     BASIC START OF SOURCE PROGRAM
08020           PRGEND  EQU   $8020H    BASIC END OF SOURCE PROGRAM
A100            COLDS   EQU   $A100     BASIC COLD ENTRY
A103            WARMS   EQU   $A103     BASIC WARM ENTRY
08217           VRBBEG  EQU   $8217     BASIC START OF VERB TABLE
08361           VRBEND  EQU   $8361     BASIC END OF VERB TABLE
                *****************************************************
                *
0814E                   ORG   $814E
814E 27 D9              FDB   NEW       NEW START OF SOURCE PROGRAM AFTER BASIC
                *
8240                    ORG   $8240     OLD LET VERB ADDRESS
8240 46                 FCC   /FIN/     REPLACE LET WITH FIND VERB
8250 00                 FCB   0
8251 26 A5              FDB   START     ADDRESS OF FIND ROUTINE
                *****************************************************
                *
2690                    ORG   $2690
                *****************************************************
                *
2690            VERB    RMB   2
2692            BUFF    RMB   2
2694            TXTTMP  RMB   2
2696            BUFTMP  RMB   2
2698            LN      RMB   2
269A            LINBEG  RMB   2
269C            FLAG    RMB   1
                *****************************************************
                *     TERMINAL PROMPT TO ENTER WORD(S) TO FIND
                *****************************************************
269D CE 27 D6   START   LDX   #PROMPT   POINT TO MESSAGE
26A0 BD AD 1E           JSR   PSTRNG    PRINT IT
                *
                *****************************************************
                *     ENTER ON THE TERMINAL THE WORD(S) YOU WANT TO FIND
                *****************************************************
26A3 CE A0 00   INPUT   LDX   #BUFFER   STORE DATA FROM TERMINAL HERE
26A6 BD AD 1B   INPUT1  JSR   INBUFF    GET A CHARACTER FROM THE TERMINAL
                *
                *****************************************************
                *          SEE IF THE INPUT WAS A VERB
                *****************************************************
26A9 CE 82 17   VERB1   LDX   #VRBBEG   GET THE START OF THE VERB TABLE
26AC FF 26 90           STX   VRB
26AF CE A0 00           LDX   #BUFFER   GET THE START OF THE BUFFER INPUT
26B2 FF 26 9A           STX   BUFF
26B5 FE 26 9A   VERB2   LDX   BUFF      POINT TO THE BUFFER CHARACTER
26B8 E6 00              LDA B 0,X       GET THE CHARACTER
26BA C1 00              CMP B #$00      END OF BUFFER WORD?
26BC 27 31              BEQ   VERB4     YES, SO GET THE VERB ADDRESS
26BE FE 26 90           LDX   VERB      NO, SO POINT TO THE VERB CHARACTER
26C1 BC 03 61           CPX   #VRBEND   IS THIS THE END OF THE VERB TABLE?
26C4 27 30              BEQ   LINO1     YES, SO TRY TO FIND A MATCH IN THE TEXT
26C6 A6 00              LDA A 0,X       NO, SO GET THE VERB CHARACTER
26C8 11                 CBA             DO THE TWO CHARACTERS MATCH?
26C9 27 17              BEQ   VERB5     YES, THEY MATCH
26CB 01 00      VERB3   CMP A 00        NO, GO GET PAST THIS VERB
26CD 27 05              BEQ   VERB4
26CF 08                 INX
26D0 A6 00              LDA A 0,X
26D2 20 F7              BRA   VERB3
26D4 08         VERB4   INX             POINT TO NEXT VERB
26D5 08                 INX
26D6 08                 INX
26D7 FF 26 90           STX   VERB
26DA CE A0 00           LDX   #BUFFER   GET THE START OF THE BUFFER BACK
26DD FF 26 9A           STX   BUFF
26E0 20 D3              BRA   VERB2
26E2 08         VERB5   INX             GET NEXT VERB CHARACTER
26E3 FF 26 90           STX   VERB      SAVE THE LOCATION
```

```
26E6 FE 26 9A           LDX   BUFF      GET THE BUFFER
26E9 08                 INX             GET THE NEXT BUFFER CHARACTER
26EA FF 26 9A           STX   BUFF      SAVE THE LOCATION
26ED 20 C6              BRA   VERB2
26EF FE 26 90   VERB6   LDX   VRB       GET THE VERB ADDRESS
26F2 08                 INX
26F3 FF 26 90           STX   VERB      THIS IS THE ADDRESS OF THE VERB
                *****************************************************
                *     TRY TO MATCH TERMINAL INPUT WITH CODED VERB
                *****************************************************
26F6 DE 2E      FIND1   LDX   PRGBEG    POINT TO PROGRAM START
26F8 B6 26 A4   FIND2   LDA A FLAG      SEE IF PROGRAM END WAS REACHED
26FB 81 FF              CMP A #$FF
26FD 27 23              BEQ   FIND4     YES, THE END WAS REACHED
26FF FF 26 A0           STX   LN        NO, SO SAVE THE LINE NUMBER
2702 08                 INX             GET TO THE START OF THE
2703 08                 INX             TEXT IN THIS LINE
2704 08                 INX
2705 08                 INX
2706 08                 INX
2707 FF 26 A2           STX   LINBEG
270A 08                 INX
270B 9C 20              CPX   PRGEND    IS THIS THE END OF THE PROGRAM?
270D 26 05              BNE   FIND3     NO, SO KEEP ON LOOKING
270F 86 FF              LDA A #$FF      YES, SO PREPARE TO END
2711 97 26 A4           STA A FLAG
2714 09         FIND3   DEX             GET BACK TO THE VERB ADDRESS
2715 09                 DEX
2716 09                 DEX
2717 EE 00              LDX   0,X       GET THE VERB ADDRESS
2719 BC 26 90           CPX   VERB      IS THIS THE VERB WE WANT?
271C 26 17              BNE   TEXT1     NO, GO TRY TO FIND A MATCH IN THE TEXT
271E 7E 27 09           JMP   PLINE1    YES, SO GO AND PRINT THE ENTIRE LINE
2721 BE 27 CF   FIND4  LDX   #NONE     WE WILL END THIS ROUTINE, NOW
2724 BD AD 1E           JSR   PSTRNG
2727 86 00              LDA A #$00      RESET FLAG
2729 B7 26 A4           STA A FLAG
272C CE 00 00           LDX   #$00      RESET VERB
272F FF 26 90           STX   VERB
2732 7E 01 03           JMP   $0103     BACK TO BASIC
                *****************************************************
                *          TRY TO MATCH TERMINAL INPUT WITH TEXT
                *****************************************************
2735 CE A0 00   TEXT1   LDX   #BUFFER   POINT TO THE WORD TO FIND
2738 FF 26 9E           STX   BUFTMP    SAVE THE LOCATION
273B FE 26 A2   TEXT2   LDX   LINBEG    SAVE THE START OF THE TEXT LINE
273E FF 26 9C   TEXT3   STX   TXTTMP    SAVE OUR PLACE IN THE TEXT
2741 A6 00              LDA A 0,X       GET A CHARACTER FROM THE TEXT
2743 81 00              CMP A #$00      IS THIS THE END OF THE TEXT LINE?
2745 27 14              BEQ   TEXT4     YES, GO LOOK AT THE NEXT LINE
2747 FE 26 9E           LDX   BUFTMP    NO, SO POINT TO THE WORD TO FIND
274A E6 00              LDA B 0,X       GET A CHARACTER FROM THE BUFFER
274C 11                 CBA             IS IT A MATCH?
274D 27 0F              BEQ   TEXT5     YES, GO TRY TO MATCH ANOTHER BYTE
274F CE A0 00           LDX   #BUFFER   EITHER NO MATCH OR INCOMPLETE MATCH
2752 FF 26 9E           STX   BUFTMP    GO RESET BUFFER POINTER
2755 FE 26 9C           LDX   TXTTMP
2758 08                 INX
2759 20 E3              BRA   TEXT3     GO AND LOOK AT THE NEXT TEXT BYTE
275B 08         TEXT4   INX             POINT TO THE NEXT LINE NUMBER
275C 20 99              BRA   FIND2     GO AND LOOK AT THE NEXT LINE
275E 08         TEXT5   INX             POINT TO THE NEXT BUFFER CHARACTER
275F E6 00              LDA B 0,X
2761 C1 00              CMP B #$00      WAS ENTIRE BUFFER WORD MATCHED
2763 26 12              BNE   TEXT6     NO, IT WAS NOT
2765 BD 27 C2           JSR   LINE      IT WAS, SO PRINT THE LINE NUMBER
2768 BD 27 C9           JSR   SPACE     AND A SPACE
276B FE 26 A2           LDX   LINBEG    GET TO THE START OF THIS LINE
276E 09                 DEX             GET BACK TO THE VERB ADDRESS
276F 09                 DEX             TO ALLOW PRINTING OF THE VERB
2770 09                 DEX
2771 EE 00              LDX   0,X
2773 09                 DEX
2774 7E 27 93           JMP   PLINE3    PRINT THE REST OF THE LINE
2777 FF 26 9E   TEXT6   STX   BUFTMP    SAVE OUR PLACE IN THE BUFFER
277A FE 26 9C           LDX   TXTTMP    POINT TO OUR PLACE IN THE TEXT
277D 08                 INX             GET TO THE NEXT CHARACTER
277E 20 BE              BRA   TEXT3     GO AND LOOK FOR ANOTHER MATCH
                *****************************************************
                *                PRINT THE ENTIRE LINE
                *****************************************************
2780 BD 27 C2   PLINE1  JSR   LINE      PRINT THE LINE NUMBER
2783 BD 27 C9           JSR   SPACE     PRINT A SPACE
2786 FE 26 90   PLINE2  LDX   VERB      POINT TO THE VERB
2789 09                 DEX             GET TO THE VERB ASCII
278A 09         PLINE3  DEX
278B A6 00              LDA A 0,X
278D 81 00              CMP A #$00
278F 26 F9              BNE   PLINE3
2791 08         PLINE4  INX             THIS IS THE VERB ASCII
2792 08                 INX
2793 08                 INX
2794 A6 00      PLINE5  LDA A 0,X       GET THE ASCII IN ACC A
2796 81 00              CMP A #$00      IS THIS THE END OF THE ASCII VERB?
2798 27 06              BEQ   PLINE6    YES IT WAS SO PRINT THE REST OF THE LINE
279A BD AD 18           JSR   PUTCHR    NO, SO KEEP ON PRINTING THE VERB
279D 08                 INX
279E 20 F4              BRA   PLINE5    GET THE NEXT ASCII CHARACTER
27A0 BD 27 C9   PLINE6  JSR   SPACE     PRINT A SPACE
27A3 FE 26 A2           LDX   LINBEG    POINT TO BEGINNING OF TEXT
27A6 A6 00      PLINE7  LDA A 0,X       GET THE CHARACTER
27A8 81 00              CMP A #$00      IS THIS THE END OF THIS LINE?
27AA 27 06              BEQ   PLINE8    YES, SO GET TO THE NEXT LINE
27AC BD AD 18           JSR   PUTCHR    NO, SO PRINT THE CHARACTER
27AF 08                 INX             GET THE NEXT CHARACTER IN THE LINE
27B0 20 F4              BRA   PLINE7
27B2 08         PLINE8  INX             GET TO THE START OF THE NEXT LINE
27B3 BD AD 24           JSR   PCRLF
27B6 7E 27 80           JMP   FIND2
                *****************************************************
27B9 FE 26 A0   LINE    LDX   LN        GET THE LINE NUMBER
27BC BD AD 45           JSR   OUTADR    PRINT IT
27BF 39                 RTS
                *****************************************************
27C0 86 20      SPACE   LDA A #$20      SET UP A SPACE
27C2 BD AD 18           JSR   PUTCHR    PRINT IT
27C5 39                 RTS
```

```
**************************************************************
270F 42        NONE    FCC    /BOTTOM/
2705 04                FCB    4
**************************************************************
2706 3F        PROMPT  FCC    /? /
2708 04                FCB    4
**************************************************************
2709           NEW
                       END    COLD8
     NO ERROR(S) DETECTED
```


PRESS RELEASE

47MB HIGH PERFORMANCE WINCHESTER DRIVES

For use with GIMIX versions of OS9 or UniFLEX.
Average access time is 30 milliseconds.

5MB REMOVEABLE PAK WINCHESTER DRIVES

For use with GIMIX versions of FLEX or OS9. Can be used as back-up or for systems not requiring larger disk Winchesters. 1/2 height size allows GIMIX systems to be configured with 1 47MB or 1-5MB Winchester, 1 Floppy, and 1 or three.

Two addr use times are now in stock. GIMIX mainframe systems can be ordered using Winchesters and/or Floppies. Existing GIMIX systems owners can upgrade.

Contact GIMIX to customize your system to your needs.

1925 N. Mistletoe
San Antonio, TX 78201
1 October, 1983

Mr. Don Williams
Editor, '68' Micro Journal
5900 Cassandra Smith
PO Box 849
Hixson, TN 37343

Dear Mr. Williams:

We would like to announce the formation of a 6809 User's Group in San Antonio, Texas. Our group is small, at present, but we are interested in both hardware and software aspects of the 6809 microprocessor. We plan to meet informally about once a month, and all interested persons are welcome to attend. For more information, please contact Dave Lapointe, after 7:00 p.m. at 512-732-6876 (voice).

Sincerely,
Dave Lapointe et al.

John H. Deal
122 Moorings Park Drive Apt. 709
Naples Florida 33942
1-813-261-0536

September 26, 1983
page No. 1

68 MICRO JOURNAL
5900 Cassandra Smith
P.O. Box 849
Hixson Tennessee 37343

ATT: Mr. Don Williams Sr., Publisher

Dear Don:

As you probably remember I sent you a program called BASLIST, which you accepted for input in your excellent JOURNAL. Since that time I have made some improvements in the program. As now written, the program allows user selection of LEFT MARGIN, and will automatically indent the wrap around portion of a line properly, for line numbers of up to 5 digits. Also the number of wrap around lines is unlimited, depending only on the length of a line permitted by BASIC. It does come into play, however, for large left margins.

I am enclosing a copy of the improved program in both the normal 10 C.P.I. and condensed 16.5 C.P.I. modes. Note that the left margin is set at 25 in the 16.5 C.P.I. mode. This may be changed to suit by altering 'X' in line 560.

I am also enclosing a copy of the program for the RS COLOR COMPUTER in the event you may want to use it also. It has all the attributes of the FLEX version except for the absence of the condensed mode.

In the hope that you have not type set the version of the program which I sent earlier, please destroy it and use this improved version.

Sorry for any confusion that all this may cause you Don, but I thought it worthwhile to send these new programs along.

I am using an OKIDATA 83A printer. CHR$(29) and CHR$(30) print 16.5 and the normal 10 C.P.I. respectively. CHR$(29);CHR$(31) print Bold Type.

Very truly yours,

John H. Deal

John H. Deal

```
SEPTEMBER 23 1983          FLEX 9.0 - DISK NO. 1000          BASLIST.BAS
JOHN H. DEAL                                                 PAGE NO. 1

100 REM .. THIS PROGRAM CALLED ---------- ' B A S L I S T '
110 EXEC,"TTYSET DP=0,WD=0"
120 PRINT CHR$(12)
130 PRINT TAB(25);"BASLIST"
140 PRINT TAB(20);"FOR FLEX 9.0 XBASIC":PRINT:PRINT
150 PRINT"THIS PROGRAM WILL LIST A DISK BASIC FILE. IT WILL PRINT A
    HEADER AND ALLOWS YOU TO SET THE LEFT MARGIN."
160 PRINT"IT WILL ALSO INDENT THE WRAP AROUND PORTION OF A LINE 1 TO 5
    SPACES DEPENDING ON THE NUMBER OF DIGITS IN THE LINE NUMBER"
170 PRINT
180 REM ***** ***** *****
190 INPUT"WHAT IS THE BASIC PROGRAM TO BE LISTED (AS 'BASLIST')";B$:PRINT
200 B$=B$+".BAS":P=0:C=2
210 INPUT"WHAT IS THE DISK NUMBER (AS 1000)";F:PRINT
220 INPUT"DO YOU WANT LISTING AT 16.5 CPI (Y/N)";K$:PRINT
230 INPUT"WHAT IS TODAYS DATE (AS JUNE 12 1983)";D$:PRINT
240 INPUT"WHAT LEFT MARGIN DO YOU WANT (AS 5)";X:PRINT
250 REM ***** ***** *****
260 OPEN "O",PRINT" AS 0
270 GOSUB 440
280 IF LEFT$(K$,1)="Y" THEN GOTO 540
290 ON ERROR GOTO 410
300 OPEN OLD B$ AS 1
310 INPUT LINE #1,L$
320 FOR N=2 TO 6:IF MID$(L$,N,1)()" " THEN NEXT N
330 L1=N
340 IF LEN(L$)>(80-X) THEN GOSUB 710:GOTO 310
350 PRINT #0,TAB(X);L$
360 IF C=53 THEN PRINT #0, HR$(12);:GOSUB 440:C=0:GOTO 310
370 C=C+1
380 GOTO 310
390 EXEC,"TTYSET DP=58"
400 END
410 IF ERR()8 THEN ON ERROR GOTO 0
420 RESUME 390
430 REM ***** ***** *****
440 REM .. SUBROUTINE TO PRINT HEADING
450 C=0:P=P+1
460 P$="PAGE NO."+STR$(P)
470 PRINT #0,CHR$(30)
480 PRINT #0,TAB(5);D$;
490 PRINT #0,TAB(3);;"FLEX 9.0 - DISK NO.";F;TAB(65);P$
500 PRINT #0,TAB(5);"JOHN H. DEAL";TAB(65);P$
510 REM
520 PRINT #0:PRINT #0:RETURN
530 REM ***** ***** *****
540 REM .. SUBROUTINE FOR 16.5 CPI
550 ON ERROR GOTO 600
560 PRINT #0,CHR$(29);X=25
570 OPEN OLD B$ AS 1
580 INPUT LINE #1,L$
590 FOR N=2 TO 6:IF MID$(L$,N,1)()" " THEN NEXT N
600 L1=N
610 IF LEN(L$)>(125-X) THEN GOSUB 860:GOTO 580
620 PRINT #0,TAB(X);L$
630 IF C=51 THEN PRINT #0,CHR$(12);:GOSUB 440:PRINT #0,CHR$(29)
640 C=C+1
650 GOTO 580
660 EXEC,"TTYSET WD=81,DP=58":PRINT #0,CHR$(30);:CLOSE 1:CLOSE 0
670 END
680 IF ERR(18 THEN ON ERR R GOTO 0
690 RESUME 660
700 REM ***** ***** *****
710 REM .. SUBROUTINE TO INDE T THE WRAP AROUND LINE
720 GOSUB 890
730 PRINT #0,TAB(X);L1$;C=C+1
740 L2$=RIGHT$(L$,(LEN(L$)-LEN(L1$)))
750 IF LEN(L2$)>(80-X) THEN L$=L2$:GOSUB 800:GOTO 770
760 IF LEN(L2$)<=(80-X) THEN 780
770 PRINT #0,TAB(L1+3);L1$;C=C+1;GOTO 740
780 PRINT #0,TAB(L1+2);L2$;C=C+1
790 RETURN
800 FOR J=1 TO 20
810 L1$=LEFT$(L$,((80-X)-J))
820 IF RIGH $(L1$,1)()" " THEN 830 ELSE 840
830 NEXT J
840 RETURN
850 REM ***** ***** *****
860 REM .. SUBROUTINE FOR 16.5 CPI WRAP AROUND
870 GOSUB 970
880 PRINT #0,TAB(X);L1$;C=C+1
890 L2$=RIGHT$(L$,(LEN(L$)-LEN(L1$)))
900 PRINT"810= "L1$
910 PRINT"810- LEN(L2$)= ";LEN(L2$)
920 IF LEN(L2$)>(125-X) THEN L$=L2$:GOSUB 970:GOTO 940
930 IF LEN(L2$)<=(125-X) THEN 950
940 PRIN #0,TAB(L1+X);L1$:C=C+1:GOTO 890
950 PRINT #0,TAB(L1+X);L2$;C=C+1
960 RETURN
970 FOR J=1 TO 20
980 L1$=LEFT$(L$,((125-X)-J))
990 IF RIGHT$(L1$,1)()" " THEN 1000 ELSE 1010
1000 NEXT J
1010 RETURN
```

```
SEPTEMBER 23 1983      TRS-80 COLOR COMPUTER          BASLIST/BAS
JOHN H. DEAL                DISK NO. 1000             PAGE NO. 1

100 POKE 149,0:POKE 150,41:REM .. SETS 1200 BAUD
110 REM .. THIS PROGRAM CALLED ' B A S L I S T ' AND LISTS A BASIC
    PROGRAM SAVED IN ASCII CODE. IT WILL ALLOW SETTING A LEFT MARGIN, AND
    WILL INDENT THE WRAP AROUND PORTION OF A LINE 1 SPACE PLUS THE NUMBER OF
    DIGITS IN THE LINE NUMBER.
120 REM ***** ***** *****
130 REM .. THE PROGRAM TO BE LISTED MUST HAVE BEEN SAVED USING THE 'A'
    FORMAT.
140 CLEAR 1000:CLS
```

```
150 INPUT"WHAT IS THE NAME OF THE BASIC PROGRAM AS (BASLIST)";A$:PRINT
160 INPUT"ENTER TODAYS DATE, AS (SEPTEMBER 18 1983)";D$:PRINT
170 INPUT"WHAT IS THE DISK NUMBER, AS (1000)";D:PRINT
180 INPUT"WHAT LEFT MARGIN, AS (4)";X
190 A$=A$:N$=N$+"/BAS":PRINT
200 C=0:P=1:GOSUB 320:PRINT
210 OPEN "I",#1,A$
220 IF EOF(1)=-1 THEN 300
230 LINE INPUT #1,L$
240 FOR N=2 TO 6:IF MID$(L$,N,1)<>" " THEN NEXT N
250 L1=N
260 IF LEN(L$)<(80-X) THEN GOSUB 480:GOTO 280
270 PRINT #-2,TAB(X);L$
280 C=C+1:IF C=58 THEN PRINT #-2,CHR$(12);:P=P+1:C=0:IF EOF(1)<>-1 THEN
    GOSUB 320 ELSE 300
290 GOTO 220
300 CLOSE #1:END
310 REM ***** ***** *****
320 REM .. SUBROUTINE TO PRINT HEADING AND SET BOLD PRINT.
330 CLS
340 PRINT #-2,TAB(X);D$;TAB(32-X);"TRS-80 COLOR COMPUTER";TAB(58+D+29);
    CHR$(31);
350 PRINT #-2,TAB(68-X);K$;CHR$(38);"/BAS"
360 PRINT #-2,TAB(X);"JOHN H. DEAL";TAB(36-X);"DISK NO.
    ";D;TAB(63-X);"PAGE NO. ";P
370 PRINT #-2:PRINT #-2:IF P>1 THEN PRINT #-2
380 C=8:RETURN
390 REM ***** ***** *****
400 REM .. SUBROUTINE TO INDENT A WRAP AROUND LINE.
410 GOSUB 490
420 PRINT #-2,TAB(X);L1$;C$;L2$
430 L2$=RIGHT$(L$,(LEN(L$)-LEN(L1$)))
440 IF LEN(L2$)>(80-X) THEN L$=L2$:GOSUB 490:GOTO 460
450 IF LEN(L2$)<=(80-X) THEN 470
460 PRINT #-2,TAB(L1+X);L1$;C$;":GOTO 430
470 PRINT #-2,TAB(L1+X);L2$;C$
480 RETURN
490 FOR J=1 TO 80
500 L1$=LEFT$(L$,(180-X)-J))
510 IF RIGHT$(L1$,1)<>" " THEN 520 ELSE 530
520 NEXT J
530 RETURN
```

AS = "" : FOR J = 1 TO X: AS = AS+" ":NEXT J
where X is the required length.

If that is not impressive enough then consider the following :-

MOVE ALL "ABCDEF" TO DATA-NAME.

This sentence will fill DATA-NAME with "ABCDEF" repetitivley until the field is full, truncating the result to fit. It can be done in BASIC but only with complicated code. Even if the TSC BASIC field definitions are used then the program is :-

BS = "ABCDEF"
FOR J = 1 TO X
BS = BS+BS
NEXT J
LSET AS = BS

where X has to be chosen such that LEN(BS) > LEN(AS).

When it comes to formatting print lines or data records then it is hard to draw a discrete veil over the capabilities of BASIC. The nearest BASIC gets is the "PRINT USING" statement and there is simply no comparison between this and the COBOL edited picture.

This is NOT intended as a diatribe against BASIC, I'm simply making the point about language design. The same or similar points could be made about other languages like PASCAL or FORTH. The language is not designed to do this sort of thing and so any attempt to force the issue results in a lot of complicated code and unwarranted assumptions.

Getting into the COBOL philosophy means dropping some of the informal ways of programming and at first it may seem slow compared to an interpreter. There is nothing slower than sitting over a hot machine writing bad code which in a few weeks is incomprehensible to the author and everyone else.

I'm sure that there are a lot of people out there who are as tired of fighting the software as I am. COBOL is not an universal panacea, it is just the best available solution for the standard data handling problems which most programs are all about, unless of course you are uninterested in processing data. In which case why have a computer?

Our COBOL implementation for FLEX has all the features of level 1 and some of the useful features of level 2 COBOL. There is a runtime package which is normally memory resident. We have included linkage features which make program development a snap. We are committed to providing support for COBOL for the foreseeable future and will be making enhancements to the compiler as time allows. Future plans include inexpensive applications packages based on our compiler. There is already a wealth of published information on COBOL and COBOL based applications, both for mainframes and micros.

Yours sincerely,

Ted Opyrchal.

COMPUSENSE LTD.
Computer Systems Consultants

P O Box 169
2860 Great Lines
London N13 5TN
Tel 01-882 0681
01-882 6936
Telex 8813271 GECOMS G

Don Williams
68 Micro Journal
Dixon
Tennessee

Our ref

Your ref

Date

Dear Don,

Finally I enclose a copy of our COBOL compiler for FLEX. This product has taken about two years to complete, and even now the manuals enclosed are preliminary versions. If you are happy with the product then the next thing I want to do is to advertise in the journal.

A lot of circumstances have conspired to delay this software. Mainly the time taken in writing the manuals. This year we have written and published a total of five software products for the DRAGON computer since January, including an assembler, monitor and full screen editor. Our software is about the most popular, of its kind, in the U.K. for the DRAGON and the assembler features in a new book, to be published in November, on DRAGON machine code by Ian Sinclair.

I guess a word or two about "why COBOL?" is in order. I suspect that my interest in FLEX and the 6809 is about as passionate as most. Many buffs are into machine related performance and this is similar to the early days of mainframe, witness the number of articles in the 68 Journal comparing the performance of various processors. Trouble is, when I actually want to write a useful program then I'm not interested in performance at the expense of usability or fancy structured syntax without precise control of DATA.

Like many people I tried precompiler BASIC and although this is a vast improvement over other methods of programming there is no significant improvement in language function. BASIC, or for that matter PASCAL FORTH C etc., is incapable, by DESIGN, of most of the standard D.P. functions which I need when writing programs. For example string handling verbs in BASIC are a bolt on extra which were invented to circumvent the imprecise way that BASIC defines its variables.

COBOL is limited for its syntax, verbosity, structure and the color of the paper it uses. In fact I have rarely met a buff who likes COBOL at all. On the other hand most programmers who have to produce functional applications software regularily use COBOL. There is a good reason for this, COBOL was designed for data processing and over years it has proven itself to be capable of just that. The availability of a COBOL compiler means a commercial acceptance of the capabilities of a computer or operating system and it is high time that this facility was available for FLEX.

COBOL scores for its elegance in data definition, and it has a well defined language structure purposely designed to handle meaningful record structures. As a result most programs, and programmers, only need to use a minimum of program syntax to achieve results.

For example to clear out a data field to spaces one simply says :-

MOVE SPACES TO DATA-NAME.

in BASIC this would be :-

structured constructs and other concepts intended to assist in the development of programs embodying such desirable traits as structure and modularity. PL/9 is a true compiler, generating 6809 machine code, without the use of an intermediate assembly language stage.

The A/BASIC language is a version of BASIC without many high-level constructs often found in general purpose versions of BASIC. It also has several lower-level constructs not normally found in other versions of BASIC. Although it has no structured constructs as does PL/9, it does have string processing and input/output constructs supporting terminal and disk files, unlike PL/9, which has no string or input/output constructs, and thus may be logically classified as a high-level language. A/BASIC is a true compiler generating 6809 machine code, without the use of an intermediate assembly language stage. A/BASIC is also available in a version for the 6800, although several features (such as random disk I/O) are available only in the 6809 version.

DOCUMENTATION

The product documentation is the first representation of a new new product that a prospective buyer investigates or a new user normally studies. It is important for the manual to convey a sense of the product's worth to the reader. Both A/BASIC and PL/9 have problems with their manuals which could be corrected if the proper emphasis were placed on documentation.

The PL/9 manual is far superior to the A/BASIC manual. It has a table of contents, lacking in the latter. It is lengthier, easier to read, and contains more detailed descriptions and examples than are in the A/BASIC manual. Neither has an index nor quick reference syntax guide, although quick refwrence guides are provided at the end of this article.

Neither has an introductory section to introduce a newcomer to the language. Both manuals are rather disorganized and make it rather difficult to find some specific points, such as statement, operator, and function syntax and semantics. However, the PL/9 manual has a very detailed description of the internal specifications of the PL/9 compiler.

The PL/9 manual appears much more professionally done than the A/BASIC manual. It is prepared on a word processor and the A/BASIC manual is manually prepared and has strange vertical and horizontal spacing, probably due to multiple levels of revisions, additions, and deletions.

NUMBER REPRESENTATION

Both A/BASIC and PL/9 support 8 and 16 bit integer arithmetic. PL/9's 8 bit arithmetic is signed, unlike A/BASIC's, which is unsigned. Both A/BASIC and PL/9 support signed 16 bit arithmetic, although A/BASIC also supports unsigned 16 bit arithmetic.

Both languages allow the entry of numeric constants in either decimal or hexadecimal notation. Literal numbers are assumed to be in decimal notation unless preceded by a dollar symbol, in which case they are assumed to be in hexadecimal notation.

Neither language supports floating point or extended precision arithmetic directly, although both have the ability to perform both indirectly, thru the use of assembler or machine language subroutines.

The PL/9 supporting library appears on the disk is but not covered in the manual except for a mention of how to include a library member. It contains several useful packages, including a floating point library and a compatible scientific library containing the basic trigonometric and transcendental functions.

NAMING CONVENTIONS

A/BASIC uses a highly restrictive naming convention common to many BASIC processors. Variable names must start with a capital letter and may contain a second character, which may be a digit for numeric variables or "$" for character variables (strings). Numeric variables may be subscripted with one or two dimensions and character variables may be subscripted with one dimension. Character variables may be up to 32 bytes in length, unless they have a declared length, in which case they may be up to that number of bytes in length, which may not exceed 255. The first element of an array is number 1, and may be referenced with or without the (1) or (1,1) subscript. The number of elements in a subscripted variable must be a constant and cannot exceed 65535.

PL/9 uses a much less restrictive naming convention than does A/BASIC, allowing variable names to be up to 127 characters in length, all of them significant. Variable names must start with a capital letter, which may be followed by up to 126 capital letters, digits, and underlines, in any order. Since all variables must be declared, there is no need for a naming distinction to distinguish between 8 and 16 bit variables. Variables may be subscripted with one dimension, which must be a constant not exceeding 32767. The first element of an array is number 0. If a variable name is prefixed with a period, it is assumed to contain not a value but a pointer to a value, used in a manner similar to the pointer variables in "C".

DECLARATIONS

The only explicit declaration in A/BASIC is DIM, which declares a string or a numeric or string array. Variables may also be declared by use, and mis-spellings may cause new variables to be created, when the intent was to use existing variables.

PL/9 requires the declaration of all variables, avoiding A/BASIC's accidental creation of variables, but requiring somewhat more planning by the user. Variables may be declared as BYTE or INTEGER, analogous to the string and numeric variables of A/BASIC, and may be subscripted by following the declaration with the size in parentheses. For instance, the following is an example of PL/9 declarations:

```
BYTE BUFFER(80), FLAG; INTEGER COUNT, LINES;
```

Named numeric constants may be declared with the CONSTANT declaration, which is used in the following format:

```
CONSTANT CR=13, LF=10, SPACE=32;
```

One of the more powerful declarations supported by PL/9 is PROCEDURE. Its closest A/BASIC analog is a subroutine which is the target of a GOSUB. It establishes a named subroutine with or without parameters and with or without local variables. Any parameters are enclosed in parentheses immediately after the procedure name and are separated by colons. The declarations of any local variables immediately follow the parameter list. Parameters and local variables are declared as BYTE or INTEGER and may be subscripted with one dimension only.

Executable code in a PL/9 program must be contained in one or more PROCEDUREs. Each PROCEDURE may or may not return a value, and each call must be consistent with the declaration of the PROCEDURE. If a PROCEDURE is to return a value, its type (INTEGER or BYTE) is determined from the type of the expression returned by a RETURN or ENDPROC statement. Although it is not stated in the manual, the last PROCEDURE declared is assumed to contain the main program code. An alternate form of PROCEDURE is ASMPROC, which introduces a machine language subroutine, but is otherwise declared and used in a manner similar to PROCEDURE.

Variables declared outside of any PROCEDUREs are global, and the word GLOBAL must precede the BYTE or INTEGER declaration. Only one GLOBAL statement is allowed per program unit. Global variables are accessable in all subsequent PROCEDUREs. However, PROCEDURE parameters, local variables, and labels may have the same names as global variables, in which case the local declarations override the global declarations temporarily.

Labels are declared by usage within a procedure, and are local to the PROCEDURE of declaration. They are used only as the destination point of GOTO statements. They must conform to the naming conventions of other identifiers. The actual destination point is designated by a label followed by a colon.

The PL/9 symbolic debugger requires a table containing the names, addresses, lengths, and types of all variables used in the program. Thus the compiler restricts the user to 128 each of the following named variables:

```
procedures
globals
locals and parameters per procedure
labels per procedure
named constants
data statements
```

These restrictions will tend to limit the complexity of the programs processed with PL/9, although the typical program for dedicated applications is not very complex, in terms of the number of variables and labels required.

MEMORY ALLOCATION

Both PL/9 and A/BASIC have default and specified methods of allocating space for data, program, and stack areas.

By default, A/BASIC allocates variables starting at $8000 up and PL/9 allocates variables starting at the end of the stack down, with PROCEDURE parameters and local variables using temporary locations at lower addresses than the fixed global variables. A/BASIC uses a statement of the following form to modify the current data pointer, and thus subsequent memory allocation:

```
BASE=address
```

PL/9 prefixes a variable declaration with a sequence of the following form to modify allocation for that declaration only:

```
AT address;
```

By default, A/BASIC starts program code at address $1000 and PL/9 places program code as high as possible in memory; PL/9 code is position-independent, so this is possible. A/BASIC uses the following statement to modify subsequent program code allocation:

```
ORG=address
```

PL/9 uses the following statement to modify subsequent program code allocation:

```
ORIGIN=address
```

However, origin statements are ignored by the PL/9 symbolic debugger, which loads program code as high as possible in memory.

By default, A/BASIC and PL/9 use the stack space allocated by the monitor or operating system. Both use a statement of the following form to set the initial stack pointer:

```
STACK=address
```

PL/9 allows a statement of the following form to set the initial stack pointer to the start of the program code, assuming the statement is placed before any PROCEDUREs:

```
STACK=*
```

compatible compiler conforms to the Kernighan and Ritchie C specification including long, float, and double data types.

Because of the similar characteristics of the OS-9 and Unix operating systems, the C compiler provides source code level compatibility so that application software written for Unix can be also be run on OS-9 or the reverse. The output of the compiler is optimized 6809 assembly language source code which can be run on OS-9 or used in stand-alone systems such as ROM-based control systems.

A unique feature of the Microware C compiler is its real-time profiler capability. When activated, the profiler counts procedure invocations during program execution. A report printed after the program runs gives a statistical breakdown of function execution frequency. Using this information the programmer can identify which functions can most profitably be optimized.

A relocating macro assembler, linkage editor, and comprehensive standard function library is included in the compiler package. The standard library includes all C standard functions plus Unix and OS-9 system calls. Documentation consists of a comprehensive User's Manual plus a copy of the Kernighan and Ritchie book "The C Programming Language".

The C compiler package is available now from manufacturers and distributors of OS-9 based computers or directly from Microware.

"JUST" REVIEW

REVIEW OF JUST TEXT FORMATTER

This is a review of a new (yet another) text processor. It is dedicated to the Epson printer or any command compatible facsimiles. The name of the new formatter is JUST and it is available in the Flex operating system. It was written by Ronald W. Anderson, the talented author of Flex Users' Notes, a monthly column in this journal. The term, JUST, refers to "justify", the placement of the textline so that it is aligned with the other lines on the left of right margins or when needed, the center of the line.

JUST has most of the standard commands. They are invoked as follows:

,W:nn	Set width of line too nn characters
,M:nn	Set left margin nn spaces
,P	Starts a new paragraph
,S:nn	Skip nn lines
,J	Turns on justify mode
,Q	Turns off justify mode

There are, of course, many others. There are some commands that are unique and these form the most interesting facet of the new program. These are:

,E	Sets emphasized print mode on the Epson printer
,D	Sets double strike mode
,B	Sets both, for boldface characters
,N	Cancels the other Epson commands

On Epson printers equipped with a graphics ROM, (Graftrex), there are provisions for inserting commands in the middle of text for cause special printing for less than an entire line. These include:

\0	Turn on italics
\1	Turn off italics
\2	Boldface (double strike) on
\3	Boldface (double strike) off

These commands are embedded within the text. For example, \0everything from here to here\1 will be italicized. This example will \2be printed in boldface\3.

These commands can also be \0\2be printed in boldface italics\1\3.

The package includes the program source in a new compiler called PL9. There is also a short segment of assembled code containing the code for the Epson commands. The instruction manual is present as a textfile with processor commands. JUST can be tried out by inserting the disk and typing:

```
L.JUST.CMD L.JUSTINST.TXT T
```

This will cause the instructional manual to be directed to the terminal. "P" can be used to send the processed file to the printer. Instead of "P" or "T" a file name can be used and the file will be saved in a processed form on the disk. When text is directed to a diskfile the printer commands are included. These, however, cannot be sent to the printer by the FLEX LIST.CMD routine which does not transmit characters with ASCII values below $20. Epson commands utilize ESCAPE ($1B). To get around this problem, a short utility file is included in the package.

This program, FPRINT.CMD, will send all characters to the printer and allow the text file to properly configure it. This works fine except for $09 which is interpreted as Horizontal Tab by FLEX.

Thus far, I have described an interesting exercise in the study of text formatters. What is unique about JUST is that it is written in PL9 and the source is included. For we novices in PL9 (and structured programming in general), JUST is a superb entry into the use of PL9 to devise large programs of practical use. The source for JUST can be easily modified with any text editor and recompiled quickly for the purpose of studying PL9 or of making modifications and/or improvements in the program.

Perhaps it will serve as a vehicle to allow full utilization of the Epson printer with Graftrax option. There are already software packages in CP/M offering large numbers of character fonts for use on the Epson. The current version of JUST hopefully is just the beginning of a many refinements and additions. I enjoy using JUST and look forward to experimenting with the source.

Theodore A. Peintuch, M.D.

WESTCHESTER Applied Business Systems
Post Office Box 187, Briarcliff Manor, N.Y. 10510

PRODUCT NEWS
October, 1983

XDMS V1.1

XDMS V1.1 is the first major revision to the XDMS Data Management System. This version incorporates several enhancements designed to add processing flexibility within the GENERATE process control statements. For example, it is now possible to do rate table lookups (eg: tax, FICA, etc) on ranges of values and incorporate the found values in subsequent calculations. Immediate commands have been added to permit in-process file deletes or purge, terminal messages and prompts, and optional escape from the process. GENERATE error handling is now non-fatal, allowing interactive users to correct or reenter instructions without reloading the program. XDMS V1.1 replaces V1.0 effective October 1, and is available for $179.95. Upgrades for registered XDMS V1.0 users is $29.00.

XDMS+

XDMS+ adds to V1.1 a set of useful utilities. SET permits display or change of the XDMS margin, width, page, ejects, backspace, end-of-line, delete, return echo, escape, reprint, null count, pause, backup, work drive and date values. PLOT produces horizontal bar, symbol or time charts. COPYDEF defines new files with existing formats. REVISE permits modification of file field definitions. OUTPUT dumps a .DMS file in tabular ASCII format. INPUT loads a .DMS file from dump output or user generated text. FORM displays the print form of a .DMS file. SIZE lists the number of file records and sectors. PURGE erases the data portion of a .DMS file. FILES displays a file name matrix for a given file extent. REDEFINE allows change of a field name. The utilities are available seperately to new XDMS V1.1 users for 79.95 - However, since we value our existing customer base, we are extending a $30.00 one time special to XDMS V1.0 users to purchase both XDMS V1.1 AND the utilities. This offer will expire December 1, 1983.

XACC

The XACC General Accounting System has also been redesigned under V1.1. The system now permits optional random access for the ACCOUNTS and PRODUCTS files, a user defined ORDER file, custom invoicing (similar to GENERATE's form capacity) and a percentage split of a transaction amount over two accounts. The aggregation process in BALANCE, CLOSE, LEDGER and INCOME has been redesigned and is about five times faster. The product code field may now be alphanumeric if desired. XACC V1.1

(and XDMS V1.1) are designed to run with "narrow" screens of 30-64 characters if necessary (eg; the Color Computer). We have submitted a "V64" CoCo screen to South East Media for inclusion in the F-MATE pkg. This uses a 3-Pixel,1-Space character format and we will supply the module free with any order - If you assure us that you have purchased the F-MATE package. The new XACC V1.1 system sells for $249.95 - However, (You guessed it) We will extend existing XACC V1.0 users the opportunity to upgrade for $25.00 (XDMS V1.1 is also required @ $25, or the optional XDMS+ V1.1 @ $50.).

MANUALS

The manuals for XDMS and XACC have been reprinted. A new cover design with color stripes and bold lettering on a white background presents a professional appearance. Additional sections have been added for the utilities, and for additional operational detail. New manuals will be shipped with upgrades.

UPGRADES

Recently, South East Media became our primary dealer. As such we pay a sales commission for orders placed there. Since we are zoned as "residential" we cannot process credit card orders directly (don't ask) and must "factor" these orders thru our dealer (and pay a commission). On upgrades, We prefer check orders placed here - They can be processed quickly and alleviate confusion as to who is a valid user. We urge new users of ANY system to place initial orders with South East Media on 800-330-6800 (toll free USA). Once a subscriber, we prefer to deal on a "direct" basis which permits us to furnish you with the latest news and modifications on a timely basis. We are continuing to negociate with our local Banks for credit card processing "privilegea". Please draw foreign checks on U.S. banks and funds - Otherwise we are charged a $15.00 fee by our bank to cash the check!

COSTS ARE UP

Right now, it looks like we might have to raise our prices in January. Commissions, royalty, advertising, printing, etc. all contribute to our costs, and our profit margin is narrowing rapidly. XDMS sales are up, but not enough to compensate. We are also considering a three package approach, by adding an entry level XDMS (simple reports only). This would be "level I" and would sell for $100-120, standard XDMS would be "level II" and sell for $180-200, and XDMS+ would be "level III" and sell for $250-300. Existing users would upgrade between levels at a discount (difference + a percent) and between versions for $25.00.

OS-9?

We are anxiously awaiting arrival of Radio Shack's OS-9, which seems to be held up in Tandy's quality control group. Rumor has it that there is a bug in the timing between disk operations and a second terminal. (Does anyone really connect a second terminal to a $400. computer?) Having studied Microware's OS-9 manual, we think that a XDMS version is feasible (but not easy). As soon as OS-9 is released, we intend to begin conversion of XDMS code. This effort should take about 3 months. The FLEX version will continue to be offered as long as a demand exists.

APPLICATIONS

We are still looking for generic XDMS applications for an applications manual. Seems like most users are using the system for very specific uses. Here, we have used it for private line network analysis and for cataloging data communications equipment - Not too applicable for a general applications manual. Any suggestions are welcome - Write Bill Adams, P.O. Box 187, Briarcliff, N.Y. 10510.

6 October 1983

Larry E. Williams, Editor
'68' MICRO JOURNAL
P.O. Box 849
Hixson, TN 37343

Dear Larry,

I guess that I am in the soup now that you published my letter to South East Media. No harm done to me, but some folks may take offense to my running a Z80 and CP/M on the 50-buss. As the old saying goes, "Since I'm already in the soup, I may as well stir."

META LAB Z809

Yes, I am running CP/M software on my Gimix 6809+. I'm doing it with a Meta Lab Z809 CPU board which I have had for longer than a year. Operationally, that Meta Lab Z809 is a fine complement to the Gimix. Even though it is one of the original Z809's, I have not had any problems with it at all. My Gimix is operating at 2 mHz, which puts the Z80 at 4 mHz, and I have never even seen so much as a dropped or garbled byte.

The way Meta Lab designed the Z809, it and the Gimix 6809+ CPU work together as dual processors with each one specializing in part of the work. The 6809 has control of the hardware and the Z80 services the application software. It's better than a marriage, neither gets in the other's way. That is, so long as I remember that the Z809 does not directly access the hardware. It hands-off all hardware instructions to the 6809. In turn, the 6809 passes all hardware specific information back to the Z809 for use in the CP/M applications programs. The data transfers are made in specially designated buffers.

BOOTING CP/M

Booting CP/M is as simple as slipping my FLEX system disk out of drive 0, sliding my CP/M system disk into drive 0, then doing a normal cold boot. CP/M comes up fast! It calls drive 0, drive A, and drive 1 is called drive B. The CP/M prompt includes the system drive letter, "A ". The Meta Lab implementation of CP/M requires that the system disks and the work disks be kept separate. They can not be interchanged between drives because the system disks have three reserved tracks and the work disks have two reserved tracks.

META LAB CP/M

The software key to this highly successful dual processor scheme is the BIOS module of CP/M. The BIOS module in CP/M does all the I/O handling, including disk I/O. Meta Lab's BIOS is written 90% in 6809 code and 10% in Z80 code. Unfortunately, Meta Lab has not been able to support their BIOS very well, so to date, it has not been particularly flexible. It was written for standard 8" single side, single density disk format and a parallel interface printer. Last winter, they did provide some patch addresses which helped me make some critical changes. But because of my experience with Gimix-FLEX and my two Datatrak-8 drives, I have been spoiled with nearly 2 meg of disk storage. Furthermore, I can now see an application coming at me which will require hard disk.

I have spent more than 100 hours laborously disassembling and interpreting Meta Lab's BIOS. The job is now about 80% complete. Without that premier disassembler, DYNAMITE+, this task would have driven me out of my mind. I now know that eventually I'll be able to take Meta Lab's BIOS all the way to a hard disk format. If I can get permission from Meta Lab, I'll be willing to share the heavily commented, though not reassemblable, results of my effort with your readers by publishing it in *'68' MICRO JOURNAL*.

CP/M vs. FLEX

Now the big issue, CP/M vs. FLEX! Will I contribute anything useful to the debate? Yes and no. If you're looking for loyal unqualified support for FLEX, I'm going to disappoint you. On the other hand, I don't rate CP/M as better. Confusing? Well I prefer to stick close to facts, especially in debates. Running both systems on the same computer

has given me an excellent opportunity to make some observations.

CP/M and FLEX are not directly comparable. CP/M is a primitive operating system with hardly any capability of its own. While FLEX is a sophisticated operating system with a full-house of features; it seems to have a life of its own. CP/M's sole purpose seems to be to support applications programs. However, quite a body of utility programs have been written for it.

With regard to system-level utilities, I have far greater respect for the ones I find associated with FLEX and the 6809. I attribute this strongly to the knowledge and skill of the people writing system software for the 6809. I will even go so far as to make the statement that there are better quality system-level programs being given away in this magazine than are available for purchase for the Z80 and CP/M. There is, however, a complete rewrite of CP/M in the public domain, called ZCPR2 (by Rich Conn). It has good reviews. I have a copy of it, but have not installed it, yet.

However superior I think 6809 system software may be, I think the opposite about applications software. I did not spend my money to buy a Z809 board because I was curious. I was forced to do so because of lack of high quality, heavy duty applications software running on FLEX. That was in the Spring of 1982. In 1983, I have been encouraged by the increase of application software advertising in *'68' MICRO JOURNAL*.

PERFORMANCE

The Meta Lab Z809 and implementation of CP/M does not seem to impair processing speed noticeably. For example it seems from Wilbur Killebrew's letter in the October issue that the 50-buss still lacks a useable FORTRAN. I also have been victimized by the FORTRAN fiasco on the 50-buss to the extent of several hundred dollars, but at an earlier time and with a different vendor. Now, Meta Lab's Z809 has enabled me to run Microsoft FORTRAN on the 50-buss. I benchmarked the now infamous Gilbreath's version of "Eratosthenes' Sieve" at 15 seconds. This is 17 times faster than Wilbur Killebrew got out of TSC's FORTRAN, and 33% faster than I clocked the same program on a PDP-11/70 (RSX-11M+ operating system, F77 compiler, no other users on the system).

VENDOR

The Z809 was last advertised in *'68' MICRO JOURNAL* in the August 1983 issue, page 59. It was priced at $595, and can be ordered from:

Meta Lab
6825 County Line Road
Longmont, CO. 80501
tel: (303)449-1711

Philip C. Nunn
201 Netherfield
Comstock Park, MI. 49321

RELINK

RELINK AVAILABLE DISK SECTORS

BY

JOSEPH D CONDON
4505 86TH STREET APT 28
DES MOINES, IOWA 50322
TELEPHONE 515-278-4581

THE FOLLOWING PROGRAM (WRITTEN IN FLEX09 ASSEMBLER) RELINKS ALL THE AVAILABLE SECTORS ON A FLEX09 FORMATTED DISK INTO ASCENDING ORDER. THIS MINIMIZES THE CHECKER BOARDING EFFECT WHICH SLOWS DOWN FILE ACCESSING TIME. THE PROGRAM BUILDS AN AVAILABLE SECTOR BIT MAP IN MEMORY WHICH IT THEN USES TO REWRITE ALL AVAILABLE SECTORS AND SECTOR LINKAGE. THE BIT MAP REQUIRES 8K OF MEMORY WHICH WILL ALLOW THE PROGRAM TO RELINK A DISK CONTAINING UP TO 65536 FREE SECTORS. THE TOTAL MEMORY REQUIREMENTS FOR THE PROGRAM IS APPROXIMATELY 9K, MOST OF WHICH IS USED FOR THE SECTOR BIT MAP. THIS PROGRAM IS NOT RELOCATABLE IN ITS PRESENT FORM. ALTHOUGH THE PROGRAM IS INTENDED TO WORK PROPERLY ON ALL TYPES OF DISK DRIVES, IT HAS ONLY BEEN TESTED ON SINGLE SIDED,SINGLE DENSITY, 35,40 AND 80 TRACK DRIVES. IF YOUR DRIVE IS NOT ONE OF THE ABOVE MENTIONED I WOULD SUGGEST USING THIS PROGRAM WITH CAUTION FOR THE FIRST TIME UNTILL YOU ARE CONFIDENT THAT THE PROGRAM WORKS PROPERLY.I HOPE THIS PROGRAM SAVES YOU AS MUCH TIME AS IT HAS ME IN THE PAST.

SYNTAX "RELINK <DRIVE NO>"

```
*                                                 *
*                                                 *
*                    *RELINK*                     *
*                                                 *
*                                                 *
*          RELINK AVAILABLE DISK SECTORS          *
*                                                 *
*                                                 *
*                    JOE CONDON                   *
*                    09/27/83                     *
*                                                 *
*                                                 *
*                                                 *
*                  FLEX EQUATES                   *
*                                                 *
```

```
0001 VERNO  EQU  1          VERSION NUMBER

0004 EOT    EQU  $04        ASCII EOT CHARACTER

CD40 FCB    EQU  $CD40      SYSTEM FILE CONTROL BLOCK
CD03 WARMS  EQU  $CD03      FLEX RE-ENTRY POINT
CD1E PSTRNG EQU  $CD1E      PRINT ASCII CHAR STRING
CD24 PCRLF  EQU  $CD24      PRINT CARR RETURN LINE FEED
CD3F RPTERR EQU  $CD3F      REPORT DISK ERROR
CD42 GETHEX EQU  $CD42      INPUT HEX VALUE
D406 FMS    EQU  $D406      FILE MANAGEMENT SYSTEM
```

```
*                                                 *
*                START OF PROGRAM                 *
*                                                 *
```

```
0000                ORG  $0000      PROGRAM ORIGIN

0000 20 01  START   BRA  RELINK     BRANCH AROUND VERSION NO
0002 01             FCB  VERNO      VERSION NUMBER
```

```
3003 BD  C042  RELNAM JSR   GETDRV  GET DRIVE NUMBER
3006 1025 2C2D        LBCS  INVDRV  INVALID DRIVE NUMBER
300A 8C  3003         CMPX  #3      TEST FOR VALID DRIVE NUMBER
300D 1027 2DCE        LBEQ  INVDRV  INVALID DRIVE NUMBER
0011 1F  10           TFR   X,D     MOVE DRIVE NUMBER TO D REG
0013 8E  CB40         LDX   #FCB    POINT TO FCB
0016 E7  03           STB   3,X     STORE DRIVE NUMBER
0018 86  30           LDA   #$30    SIR TRACK NUMBER
001A A7  88 1E        STA   $1E,X   SET FCB TRACK NUMBER
001D 86  35           LDA   #$35    SIR SECTOR NUMBER
001F A7  88 1F        STA   $1F,X   SET FCB SECTOR NUMBER
0022 86  09           LDA   #9      READ SECTOR FUNCTION CODE
0024 A7  84           STA   ,X      SET FCB FUNCTION CODE
0026 AD  9F A004      JSR   [$A004] CALL OS9 MANAGEMENT SYSTEM
002A 25  26 00C1      LCS   DISKER  DISK ERROR
002E EC  88 50        LDD   $50,X   GET BEGINING FREE ADDRESS
0030 FD  3058         STD   FREBEG  STORE BEGINING FREE ADDRESS
0033 EC  88 5F        LDD   $5F,X   GET ENDING FREE ADDRESS
0036 FD  015C         STD   FREEND  STORE ENDING FREE ADDRESS
0039 DE  0160         LDU   #SWAP   POINT TO DRIVE SECTOR MAP

003C 6F  80    RELL   CLR   ,X+     CLEAR SECTOR MAP
003E 8C  2160         CMPX  #$SWAP+$2000 TEST FOR END OF MAP
0041 26  F9           BNE   RELL    CLEAR NEXT MAP LOCATION
0043 8E  CB40         LDX   #FCB    POINT TO FCB
0046 EC  8854         LDD   FREBEG  GET FREE BEGINING ADDRESS

0049 1083 2000 REL2   CMPD  #0      TEST FOR LAST FREE SECTOR
004D 27  19           BEQ   REL3    PREVIOUS AVAILABLE SECTORS
004F E3  88 1E        STD   $1E,X   SET FCB TRACK & SECTOR #S
0052 17  0DC5         LBSR  MAPPOS  MAKE DRIVE SECTOR MAP MASK
0055 A8  84           EORA  ,X      OR A REGISTER WITH MAP BYTE
0057 A7  84           STA   ,X      STORE MAP BYTE
0059 8E  CB40         LDX   #FCB    POINT TO FCB
005C 10  3406         SWI2  FMS     CALL FILE MANAGEMENT SYSTEM
005F 1026 3088        LBNE  DSKERR  DISK ERROR
0063 EC  88 40        LDD   $40,X   GET NEXT SECTOR ADDRESS
0066 20  E1           BRA   REL2    TEST FOR LAST FREE SECTOR

0068 8E  C080  REL3   LDX   #FCB+$40 POINT TO FCB BUFFER

006B 4F  80    RELA   CLR   ,X+     CLEAR FCB BUFFER
006D 8C  C080         CMPX  #FCB+$20 TEST FOR END OF BUFFER
0070 26  F9           BNE   RELA    CLEAR NEXT BYTE
0072 8E  CB40         LDX   #FCB    POINT TO FCB
0075 86  0A           LDA   #$0A    WRITE SECTOR FUNCTION CODE
0077 A7  84           STA   ,X      SET FCB FUNCTION CODE
0079 CE  3000         LDU   #$3000  SET BEGINING SECTOR ADDRESS
007C FD  015E         STD   SECADD  STORE SECTOR ADDRESS
007F BD  79           BSR   GETFRE  GET NEXT FREE SECTOR ADD
0081 24  25           BCC   REL7    FREE SECTOR NOT FOUND
0083 59  015C         STD   FREEND  SAVE LAST FREE SECTOR ADD
0086 ED  88 1E        STD   $1E,X   SET FCB SECTOR ADDRESS
0089 BD  9406         JSR   FMS     CALL FILE MANAGEMENT SYSTEM
008C 2A  60           BPL   DSKERR  DISK ERROR

008E EC  88 1E  REL5  LDD   $1E,X   SET SUPPORT SECTOR ADDRESS
0091 ED  88 40        STD   $40,X   SET UP NEXT SECTOR LINK
0093 BD  64           BSR   GETFRE  GET NEXT FREE SECTOR ADD
0095 24  06           BCC   REL6    NO MORE FREE SECTORS
0097 ED  88 1E        STD   $1E,X   SET FCB SECTOR ADDRESS
009A BD  9406         JSR   FMS     CALL FILE MANAGEMENT SYSTEM
009D 2A  4C           BPL   DSKERR  DISK ERROR
009F 20  EE           BRA   REL5    GET NEXT FREE SECTOR ADD

00A1 EC  88 40  REL6  LDD   $40,X   GET NEXT SECTOR ADDRESS
00A3 FD  015A         STD   FREBEG  STORE FREE BEGINING ADDRESS

00A6 86  00    REL7   LDA   #0      SIR TRACK NUMBER
00A8 A7  88 1E        STA   $1E,X   SET FCB TRACK NUMBER
00AB 86  03           LDA   #3      SIR SECTOR NUMBER
00AD A7  88 1F        STA   $1F,X   SET FCB SECTOR NUMBER
00B0 86  09           LDA   #9      READ SECTOR FUNCTION CODE
00B2 A7  84           STA   ,X      SET FCB FUNCTION CODE
00B4 BD  9406         JSR   FMS     CALL FILE MANAGEMENT SYSTEM
00B7 25  33           BCS   DSKERR  DISK ERROR
00B9 FC  015A         LDD   FREBEG  GET FREE BEGINING ADDRESS
00BC ED  88 50        STD   $50,X   STORE SIR FREE BEGINING ADD
00BE 7C  015C         INC   FREEND  GET FREE ENDING ADDRESS
00C1 ED  88 5F        STD   $5F,X   STORE SIR FREE ENDING ADD
00C4 86  0A           LDA   #$0A    WRITE SECTOR FUNCTION CODE
00C6 A7  84           STA   ,X      SET FCB FUNCTION CODE
00C8 BD  9406         JSR   FMS     CALL FILE MANAGEMENT SYSTEM
00CB 26  1E           BNE   DSKERR  DISK ERROR
00CD BD  CD24         JSR   PCRLF   PRINT CR & LF
00D0 8E  0152         LDX   #MES502 POINT TO MESSAGE
00D3 BD  CD1E         JSR   PSTRNG  PRINT MESSAGE
00D6 BD  CD24         JSR   PCRLF   PRINT CR & LF
00D9 7E  CD03         JMP   WARMS   RETURN TO FLEX

00DC BD  CD24  INVDRV JSR   PCRLF   PRINT CR & LF
00DF 8E  0132         LDX   #MES501 POINT TO MESSAGE
00E2 BD  CD1E         JSR   PSTRNG  PRINT MESSAGE
00E5 BD  CD24         JSR   PCRLF   PRINT CR & LF
00E8 7E  CD03         JMP   WARMS   RETURN TO FLEX

00EB BD  D07A  DSKERR JSR   PCRLF   PRINT CR & LF
00EE BD  CD3F         JSR   RPTERR  REPORT FMS ERROR
00F1 BD  CD24         JSR   PCRLF   PRINT CR & LF
00F4 7E  CD03         JMP   WARMS   RETURN TO FLEX

00F7 34  10    GETFRE PSH   X       SAVE X REGISTER

00F9 9C  015E  GF1    CMPX  SECADD  GET SECTOR ADDRESS
00FC 83  0001         SUBD  #1      DECREMENT SECTOR ADDRESS
00FF FD  015E         STD   SECADD  STORE SECTOR ADDRESS
0102 25  05           BMI   GF2     TEST FOR FREE SECTOR
0104 35  10           PUL   X       RESTORE X REGISTER
0106 1C  FE           ANDCC #$FE    CLEAR CARRY BIT
0108 39               RTS           RETURN FROM SUBROUTINE
```

```
010C BD  0C      GF2   BSR   MAKMSK  MAKE DRIVE SECTOR MAP MASK
010E A4  84            ANDA  0,X     TEST SECTOR MAP BIT
0110 27  EA            BEQ   GF1     FIND NEXT FREE SECTOR
0112 FC  015E          LDD   SECADD  GET SECTOR ADDRESS
0115 35  10            PUL   X       RESTORE X REGISTER
0117 1A  01            ORCC  #$01    SET CARRY BIT
0119 39                RTS           RETURN FROM SUBROUTINE

011A 34  04    MAKMSK  PSH   B       SAVE B REGISTERS
011C 44                LSRA          DIVIDE ADDRESS BY 8
011D 56                RORB
011E 44                LSRA
011F 56                RORB
0120 44                LSRA
0121 56                RORB
0122 C3  0160          ADDD  #SWAP   ADD START ADDRESS OF MAP
0125 3F  01            SWI   $1      TRANSFER D TO X REGISTER
0127 C6  80            LDA   #$80    SET MASK BIT
0129 33  84            PUL   B       RESTORE B REGISTER
012B C8  07            EORB  #$07    MASK OUT CYCLE 7

012D 58                ASLB          TEST FOR LAST SHIFT
012E 27  04            BEQ   MK2     SHIFT COMPLETE
0130 44                LSRA          SHIFT SECTOR BIT
0131 5A                DECB          DECREMENT SECTOR COUNT
0132 20  F9            BRA   MK1     TEST FOR LAST SHIFT

0134 39          MK2   RTS           RETURN FROM SUBROUTINE

0135 49 4E 56 41 MESG01 FCC   'INVALID DRIVE NUMBER.'
0139 4C 49 44 20
013D 44 52 49 56
0141 45 20 4E 55
0145 4D 42 45 52
0149 2E
014A 52 65 4C 69 MESG02 FCC   'RELINK COMPLETE.'
014E 4E 4B 20 43
0152 4F 4D 50 4C
0156 45 54 45 04
015A             FREBEG RMB   2       BEGINING FREE ADDRESS
015C             FREEND RMB   2       ENDING FREE ADDRESS
015E             SECADD RMB   2       SECTOR ADDRESS

0160             DSKMAP RMB   $2000   DRIVE SECTOR MAP

                 ********************************************

                        END   START

0 ERROR(S) DETECTED
```

CLASSIFIED ADVERTISING

**** UNIFLEX Users - SAVE $$$$ ****

New, never installed or used - High Speed SWTPC **A-1 Tape Back-up system,** 20 megabyte capacity - Complete with new DMF3 Disk Controller - Save over $1,000.00 - **ONLY $2475** Don't get caught without that **BACKUP** for all your data. **DON'T MISS THIS ONE!!** -Ask for Don, Tom or Larry, Days.

A/C 800 338-6800

***

LIKE NEW - ADDS-VIEWPOINT CRT TERMINALS - DEMO UNITS, THESE ARE LIKE NEW - WARRANTY NEVER REGISTERED - SAVE - ONLY **$478.00 EACH** - ONLY 2 LEFT. CALL ASK FOR DON, TOM or LARRY

A/C 800 338-6800

***

GIMIX II CPU+, 2 Mhz, with FLEX and OS-9 - Level 1, Ver. 1.2, latest as of 9/1/83. This CPU board is like new, we went to GIMIX III system. Set up and rea y to run. With FLEX and OS-9. Have both on your system and Just call FLEX or OS-9. Includes software and documentation. Also Time of day clock, FLEX DAT, OS-9 FPLA. Set up to require GIMIX #68 DMA disk controller. Original cost over $900.00. Special only 729.95 plus shipping (3.50). Ask for Don, Tom or Bob - 1 615 842-4601. **This will not last long!!**

***

TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. **LIKE NEW** - New cost $1295.00 - ONLY **$759.00** ready to run - Call Tom - Larry - Bob, CPI 615 842-4600

***

FOR SALE: SWTP S/00, 6809 CPU, MI9 Terminal Dual 96TPI Drives, 64k RAM, Includes software, several add-on boards. Cost new >> $8700.00. Asking $5100.00. For complete listing call 816-524-3777(work) 816-524-1123(home) or write Bill Thorsen, 200 Wingate, Lee's Summit, MO 64063.

***

Used SWTPC hardware: 8212-W Terminal $895; MP-8M 8k boards $100ea.; MP-M 4k boards $50ea.; MP-A CPU board $25; AC-30 Cassette I-O $25. Unused F&D CPU-2 and AD209-1 6809 adapter all parts except rom $150. Jonathan Meier 802-254-2285, Box 573, Brattleboro VT 05301.

FOR SALE: MSI 6800 computer system. Includes: dual 8" drives, LSI ADM-3A terminal, 2 Serial interfaces, and more. Software (BASIC compiler, and more) is included. All in good working condition. Best offer. Call or write for details. Mel Woolf, 10808 Midsummer Drive, Reston, VA 22091.

***

FOR SALE - USED TANDON 80 TRACK DS DD 5 1/4" DISK DRIVES RECONDITIONED-$250. Call 314-291-2728 - Ask for Russell.

***

New and used boards by GIMIX and SWTPC. Send SASE to: Manfred Pescke, Story Hill Rd, RFD 3, Goffstown, NH 03045

***

6809 DEVELOPMENT SYSTEM FROM CREATIVE MICRO SYSTEMS

System includes the CMS 9619 single board microcomputer with 2 MHZ clock rate, on-board I/O consisting of 2 PIA's, 2 ACIA's, one PTM, non-volatile time/date subsystem, priority interrupt vector generator, powerfail protect/restart system, and bank select outputs for memory map expansion.
System comes with CMS 20MB Winchester Disk with 8" floppy backup.
OS-9 Editor, Assembler, Debuugger, and BASIC 09 are included.
The entire system is pin and outline compatible with the Motorola Exorciser and EXORbus standard products.
CMS supports this system with a variety of memory, I/O expansion modules, controllers, etc.
The system is packaged as an attractive desk-top unit and comes with 56K static RAM.
This package was purchased from CMS a year ago for about $9,000. Our company has cancelled current development projects and we are now required to sell this system.
For more information please contact Computeer Group Supervisor at 713-932-7777, Ext. 240.

***

FOR SALE: GIMIX 6809 System. 128K RAM. 38 MEG SWTP HARD DISK. Dual DSDD 8" DISK DRIVES. Mint Condition with extremely low on-time. Have purchased another sy tem, must sell. Make offer. David Klein (201) 291-5575 after 12.

COMPILER EVALUATION SERVICES
By: Ron Anderson

The S.E. MEDIA Division of Computer Publishing Inc., is offering the following 'SUBSCRIBER SERVICE':

COMPILER COMPARISION AND EVALUATION REPORT

Due to the constant and rapid updating and enhanchment of numerous compilers, and the different utility, appeal, speed, level of communication, memory usage, etc., of different compilers, the following services are now being offered with periodic updates.

This service, with updates, will allow you who are wary or confused by the various claims of compiler vendors, an opportunity to review comparisons, comments, benchmarks, etc., concerning the many different compilers on the market, for the 6809 microcomputer. Thus the savings could far offset the small cost of this service.

Many have purchased compilers and then discovered that the particular compiler purchased either is not the most efficient for their purposes or does not contain features necessary for their application. Thus the added expense of purchasing additional compiler(s) or not being able to fully utilize the advantages of high level language compilers becomes too expensive.

The following COMPILERS are reviewed initially, more will be reviewed, compared and benchmarked as they become available to the author:

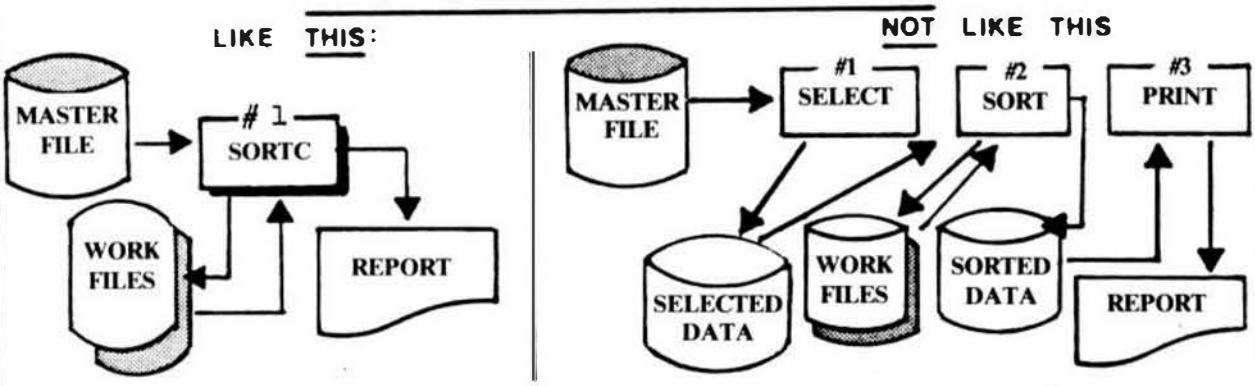
PASCAL "C" GSPL WHIMISCAL PL/9

Initial Subscription - $39.95
(Includes 1 year updates)
Updates for 1 year - $14.50

S.E. MEDIA - CPI
5900 Cassandra Smith, POB 794
Hixson, TN 37343
615 842-4601

SORTC** for OS9*

THE ONE AND ONLY



BOLDLY GOING WHERE NO SORT HAS GONE BEFORE

SORTC is a high speed, full-record compounding disk sort, which gives microcomputer users mainframe capabilities. It has been specifically designed to sort data efficiently while offering the user great flexibility in designing sort programs. It is written in BASICO9* for use under OS9.

COMPOUNDING FUNCTION

SORTC has the capability of summing user-specified numeric fields on equality of keys. This allows significant savings in memory, disk space, and program development time. A reduction in the number of disk accesses required when compared to other sorts is inherent in the design of SORTC.

DISK BASED

Specifically designed to sort large volumes of data, SORTC imposes no size restrictions on the amount of data to be sorted. It also places no limits on the number of sort keys which can be used or the order in which the keys are sorted. Furthermore, the sort procedure can be performed as many times as necessary within the same program. This feature allows the programmer to take advantage of any existing data bias, and possibly even reduce the size of the sort key.

ADVANCED DESIGN

While most disk sorts are partially based upon the Fibonacci series, SORTC is not. SORTC is a generation ahead of the normal sorts based upon the "Fib series". Its unique algorithm is automatically optimized at run time for a reduction in workspace, reduced # of disk accesses and shorter run times. Designed to be as "crash proof" as possible, the sort procedure will not abort if it is accidentally asked to sort zero items.

EASY TO USE

It is not difficult to design a program which will use JBM's SORTC. Since SORTC is a subroutine, the user may write any procedure he or she wants to format the data for sorting and then to process the sorted data. The sorted data need not be written back to disk, but instead is immediately available. The sort code is automatically inserted into the source procedure by a simple Sort Generator.

ORDERING INFORMATION

SORTC, from JBM's MIDWARE line of quality software, is available on either five and one-quarter or eight inch diskettes for a price of $150.00. All of JBM's software packages come complete with comprehensive user's manuals.

For more information, or to place an order, contact:

DEPT. FSEA
The JBM Group, Inc.
332 West Church Road
King of Prussia, PA 19406
TEL: 215-337-3138
TWX: 510-660-3999

the JBM group

VISA and MASTERCHARGE accepted.

JBM'S MIDWARE

*OS9, BASIC09 are registered trademarks of Microware Corporation.

**Uses the same algorithm as JBM's SORTC for Digital Equipment Corp. RSTS Systems.

THE BEST NEWS

DECEMBER Alford and Associates Good News Page 1983

ISAIAH 9 -

2 The people that walked in darkness have seen a great light: they that dwell in the land of the shadow of death, upon them hath the light shined.

6 For unto us a child is born, unto us a son is given: and the government shall be upon his shoulder: and his name shall be called Wonderful, Counselor, The mighty God, The everlasting Father, The Prince of Peace.

LUKE 2 -

1 And it came to pass in those days, that there went out a decree from Caesar Augustus, that all the world should be taxed.
2 (And this taxing was first made when Cyrenius was governor of Syria.)
3 And all went to be taxed, every one into his own city.
4 And Joseph also went up from Galilee, out of the city of Nazareth, into Judaea, unto the city of David, which is called Bethlehem; (because he was of the house and lineage of David:)
5 To be taxed with Mary his espoused wife, being great with child.
6 And so it was, that, while they were there, the days were accomplished that she should be delivered.
7 And she brought forth her firstborn son, and wrapped him in swaddling clothes, and laid him in a manger; because there was no room for them in the inn.
8 And there were in the same country shepherds abiding in the field, keeping watch over their flock by night.
9 And, lo, the angel of the Lord came upon them, and the glory of the Lord shone round about them: and they were sore afraid.
10 And the angel said unto them, Fear not: for, behold, I bring you good tidings of great joy, which shall be to all people.
11 For unto you is born this day in the city of David a Saviour, which is Christ the Lord.

13 And suddenly there was with the angel a multitude of the heavenly host praising God, and saying,
14 Glory to God in the highest, and on earth peace, good will toward men.

ISAIAH 53 -

1 Who hath believed our report? and to whom is the arm of the Lord revealed?
2 For he shall grow up before him as a tender plant, and as a root out of a dry ground: he hath no form nor comeliness; and when we shall see him, there is no beauty that we should desire him.
3 He is despised and rejected of men; a man of sorrows, and acquainted with grief: and we hid as it were our faces from him; he was despised, and we esteemed him not.
4 Surely he hath borne our griefs, and carried our sorrows: yet we did esteem him stricken, smitten of God, and afflicted.
5 But he was wounded for our transgressions, he was bruised for our iniquities: the chastisement of our peace was upon him; and with his stripes we are healed.
6 All we like sheep have gone astray; we have turned every one to his own way; and the Lord hath laid on him the iniquity of us all.
7 He was oppressed, and he was afflicted, yet he opened *not* his mouth: he is brought as a lamb to the slaughter, and as a sheep before her shearers is dumb, so he openeth not his mouth.
8 He was taken from prison and from judgement: and who shall declare his generation? for he was cut off out of the land of the living: for the transgression of my people was he stricken.
9 And he made his grave with the wicked, and with the rich in his death; because he had done no violence, neither was any deceit in his mouth.
10 Yet it pleased the Lord to bruise him; he hath put him to grief: when thou shalt make his soul an offering for sin, he shall see his seed, he shall prolong his days, and the pleasure of the Lord shall prosper in his hand.
11 He shall see the travail of his soul, and shall be satisfied: by his knowledge shall my righteous servant justify many; for he shall bear their iniquities.
12 Therefore will I divide him a portion with the great, and he shall divide the spoil with the strong; because he hath poured out his soul unto death: and he was numbered with the transgressors; and he bare the sin of many, and made intercession for the transgressors.

We would like to take the opportunity to thank all of our customers, associates and friends for their support over the past years, and to pray a special blessing for each and every one of you in the years to come.

******* ALFORD AND ASSOCIATES, P.O. BOX 6683, RICHMOND, VIRGINIA, 23230 - PHONE 804-320-6722 *******

TEN MOST-ASKED QUESTIONS about DYNACALC™

THE ELECTRONIC SPREAD-SHEET FOR 6809 COMPUTERS

1. What is an electronic spread-sheet, anyway?
Business people use spread-sheets to organize columns and rows of figures. DYNACALC simulates the operation of a spread-sheet without the mess of paper and pencil. Of course, corrections and changes are a snap. Changing any entered value causes the whole spread-sheet to be re-calculated based on the new constants. This means that you can play, 'what if?' to your heart's content.

2. Is DYNACALC just for accountants, then?
Not at all. DYNACALC can be used for just about any type of job. Not only numbers, but alphanumeric messages can be handled. Engineers and other technical users will love DYNACALC's sixteen-digit math and built-in scientific functions. You can build worksheets as large as 256 columns or 256 rows. There's even a built-in sort command, so you can use DYNACALC to manage small data bases — up to 256 records.

3. What will DYNACALC do for ME?
That's a good question. Basically the answer is that DYNACALC will let your computer do just about anything you can imagine. Ask your friends who have VisiCalc™, or a similar program, just how useful an electronic spread-sheet program can be for all types of household, business, engineering, and scientific applications. Typical uses include financial planning and budgeting, sales records, bills of material, depreciation schedules, student grade records, job costing, income tax preparation, checkbook balancing, parts inventories, and payroll. But there is no limit to what YOU can do with DYNACALC.

4. Do I have to learn computer programming?
NO! DYNACALC is designed to be used by non-programmers, but even a Ph.D. in Computer Science can understand it. Even experienced programmers can get jobs done many times faster with DYNACALC, compared to conventional programming. Built-in HELP messages are provided for quick reference to operating instructions.

5. Do I have to modify my system to use DYNACALC?
Nope. DYNACALC uses any standard 6809 configuration, so you don't have to spend money on another CPU board or waste time learning another operating system.

6. Will DYNACALC read my existing data files?
You bet! DYNACALC has a beautifully simple method of reading and writing data files, so you can communicate both ways with other programs on your system, such as the Text Editor, Text Processor, Sort/Merge, STYLOGRAPH™ word processor, RMS™ data base system, or other programs written in BASIC, C, PASCAL, FORTRAN, and so on.

7. How fast is DYNACALC?
Very. Except for a few seldom-used commands, DYNACALC is memory-resident, so there is little disk I/O to slow things down. The whole data array (worksheet) is in memory, so access to any point is instantaneous. DYNACALC is 100% 6809 machine code for blistering speed.

8. Is there a version of DYNACALC for MY system?
Probably. You need a 6809 computer (32k minimum) with FLEX™, UniFLEX™, or OS-9™ operating system. You also need a decent crt terminal, one with at least 80 characters per line, and direct cursor addressing. If your terminal isn't smart enough for DYNACALC, you probably need a new one anyway. The UniFLEX and OS-9 versions of DYNACALC allow you to mix different brands of terminal on the same system. There's also a special version of DYNACALC for Color Computers equipped with FLEX (Frank Hogg or Data-Comp versions).

9. How much does DYNACALC cost?
The FLEX versions are just $200 per copy; UniFLEX version $395; OS-9 version (works with LEVEL ONE or LEVEL TWO) $250. Orders outside North America add $7 per copy for postage. We encourage dealers to handle DYNACALC, since it's a product that sells instantly upon demonstration. Call or write on your company letterhead for more information.

10. Where do I order DYNACALC?
See your local DYNACALC dealer, or order directly from CSC at the address below. We accept telephone orders from 10 am to 6 pm, Monday through Friday. Call us at 314-576-5020. Your VISA or MasterCard is welcome. Please specify diskette size for FLEX or OS-9 versions. Software serial number is required for the UniFLEX version.

Order your DYNACALC today!

Computer Systems Center
13461 Olive Blvd.
Chesterfield, MO 63017
(314) 576-5020

Foreign Dealers:

Australia & Southeast Asia: order from Paris Radio Electronics, 161 Bunnerong Road (PO Box 380) Kingsford, 2032 NSW Australia. Telephone: 02-344-9111.

United Kingdom: order from Compusense, Ltd., PO Box 169, London N13 4HT. Telephone: 01-882-0681.

Scandinavia: order from Swedish Electronics hk AB, Murargatan 23-25, Uppsala 5-754 37 Sweden. Telephone: 18-25-30-00.

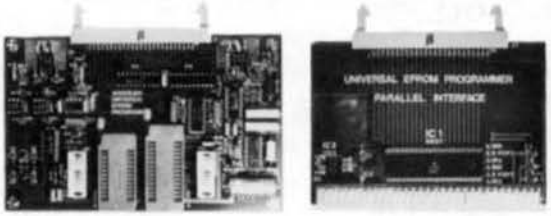
UniFLEX software prices include maintenance for the first year.

DYNACALC is a trademark of Computer Systems Center

VisiCalc is a trademark of VisiCorp.
STYLOGRAPH is a trademark of Great Plains Computer Co.
RMS is a trademark of Washington Computer Services.
FLEX and UniFLEX are trademarks of TSC.
OS-9 is a trademark of Microware and Motorola.

WINDRUSH MICRO SYSTEMS

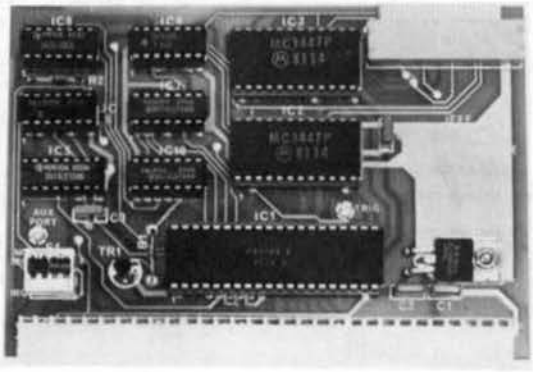
UNIVERSAL EPROM PROGRAMMER



- PROGRAMS and VERIFIES 2508, 2708, 2516, 2716, 2532, 2732A, 2564, and 2764 EPROMS. Minor hardware mods are required to program the INTEL 27128.
- Tri-volt and Single Volt 2508/2708 and 2516/2716 devices are supported.
- ZIF sockets with mode selector switches eliminate 'personality modules'.
- Twin boards with five feet of twisted pair planar cable puts the programmer out on the bench where it belongs.
- SS-30 and EXORCISOR interfaces are available.
- Menu driven software provides the following facilities:
  a. MOVE blocks of memory within the buffer.
  b. READ an EPROM into the buffer.
  c. VERIFY an EPROM against the buffer.
  d. EXAMINE and change the contents of the buffer.
  e. DUMP the contents of the buffer in HEX and ASCII.
  f. FILL a selected area of the buffer with a specified character.
- S Ftware available for all versions of $30 DOS, FLEX 2, FLEX 9 and OS-9. Assembly language source files supplied on disk....enables customizing.
- Well documented users manual provides step-by-step adaptation and operating instructions.

AVAILABLE FROM GIMIX IN THE U.S.A.

IEEE-488



- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker — Serial Poll — Single or Dual Primary Address
  - Listener — Parallel Poll — Secondary Address
  - System Controller — Group Trigger — Talk only...Listen only
- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. These drivers have been extensively tested and are GUARANTEED to work!
- Single SS-30 board (4, 8, or 16 addresses per port), fully socketed, gold plated bus connectors, and IEEE interface cable assembly.

PL/9

EDITOR/COMPILER/DE-BUGGER

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the the program a SOURCE line at a time. You also have direct access to any FLEX utility and your System Monitor.
- 250 page manual is organized as a tutorial with plenty of examples.
- Fast single pass compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files
- Signed and unsigned BYTEs and INTEGERs, 32-bit floating point REALs.
- Vectors (single dimension arrays) and Pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (%), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/.XOR).
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (CCR) and (XREG).
- FULLY supports the MC6809 SWI, SWI2, SWI3, NMI, FIRQ, IRQ and RESET vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library function modules are supplied; IOSUBS, BITIO, HARDIO, MEMIO, FLMEMIO, SCIPACK, STRINGS, and REALCON.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Anderson's FLEX User Notes column. Need we say more?

MACE/XMACE

A co-resident EDITOR/ASSEMBLER for the 6809 written by Graham Trott which takes most of the pain out of assembly language program development:

- Friendly inter-active environment where you have INSTANT access to the Editor, the Assembler, FLEX and your System Monitor.
- MACE can also produce ASMPROC's for PL/9 with the assembly language source passed to the output file as comments.
- Includes XMACE a co-resident 6800/1/2/3/8 EDITOR/CROSS ASSEMBLER.

'C'

This is the FLEX version of the James McCosh 'C' compiler that is also available on UNIFLEX from SWTP and OS-9 from Microware:

- The FLEX implementation supports the full Kernighan and Ritchie 'C' specification except 'floats', 'doubles', and 'bit-fields'.
- Produces very efficient assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%
- Supports interleaved assembly language programs.
- The TSC relocating assembler/linking loader (SPO9-17) is REQUIRED.

MACE	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)...(a steal at this price!)	$198.00
'C'	(a 56K system and the TSC SPO9-17 package is req'd)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
SS-30	Universal EPROM programmer w/one version of software	$375.00
EXORciser	Universal EPROM programmer w/one version of software	$395.00
SOFTWARE	Drivers for a 2nd, 3rd, or 4th operating system	$ 25.00

ALL PRICES INCLUDE AIR MAIL POSTAGE

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, AND ALFORD & ASSOCIATES.

FLEX (tm) is a trademark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporations, MDOS (tm) and EXORciser (tm) are trademarks of Motorola Incorporated.

An SS-50C all CMOS 256K STATIC RAM board will be available SOON!

Write for details & pricing.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.
TEL: (0692) 405189
TLX: 97380 SHARET G

FEATURES THE POWERFUL, THIRD GENERATION, MOTOROLA 6809 PROCESSOR!

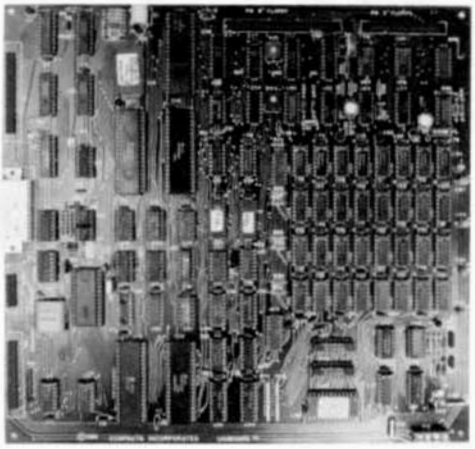
THE 6809 "UNIBOARD"™

SINGLE BOARD COMPUTER KIT

New!

PERFECT FOR COLLEGES, OEM'S, INDUSTRIAL AND SCIENTIFIC USES!

64K RAM! DOUBLE DENSITY FLOPPY DISK CONTROLLER!



BLANK PC BOARD
$99.95
WITH PAL'S, AND TWO EPROMS.

FOR 5-1/4 OR 8 INCH SOURCE DISKETTE ADD $10.

$399.00
COMPLETE KIT! FULLY SOCKETED.

ALL OPTIONS ARE STANDARD. NO EXTRAS TO BUY!

THE COMPACTA UNIBOARD™: Through special arrangement with COMPACTA INC., we are proud to have been selected the exclusive U.S. Mfg. of their new 6809 UNIBOARD™ COMPUTER KIT. Many software professionals feel that the 6809 features probably the most powerful instruction set available today on ANY 8 bit micro. Now, at last, all of that immense computing power is available at a truly unbelievably low price.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A FREE COPY IS AVAILABLE UPON REQUEST.

FEATURES:

- ★ 64K RAM using 4116 RAMS.
- ★ 6809E Motorola CPU.
- ★ Double Density Floppy Disk Controller for either 5-1/4 or 8 inch drives. Uses WD1793.
- ★ On board 80 x 24 video for a low cost console. Uses 2716 Char. Gen. Programmable Formats. Uses 6845 CRT Controller.
- ★ ASCII keyboard parallel input interface. (6522)
- ★ Serial I/O (6551) for RS232C or 20 MA loop.
- ★ Centronics compatible parallel printer interface. (6522)
- ★ Buss expansion interface with DMA channel. (6844)
- ★ Dual timer for real time clock application.
- ★ Powerful on board system monitor (2732). Features commands such as Go To, Alter, Fill, Move, Display, or Test Memory. Also Read and Write Sectors. Boot Normal, Unknown, and General Flex™.

YOUR CHOICE OF POPULAR DISK OPERATING SYSTEMS:

FLEX™ from TSC	$149
OS9™ from Microware	$199

Specify 5-1/4 or 8 inch

PC BOARD IS DOUBLE SIDED, PLATED THRU SOLDER MASKED, 11 x 11-1/2 IN.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 271-3538

TERMS: Shipments will be made approximately 3 to 6 weeks after we receive your order. VISA, MC, cash accepted. Add $4.00 shipping. USA AND CANADA ONLY

64K SS-50 STATIC RAM

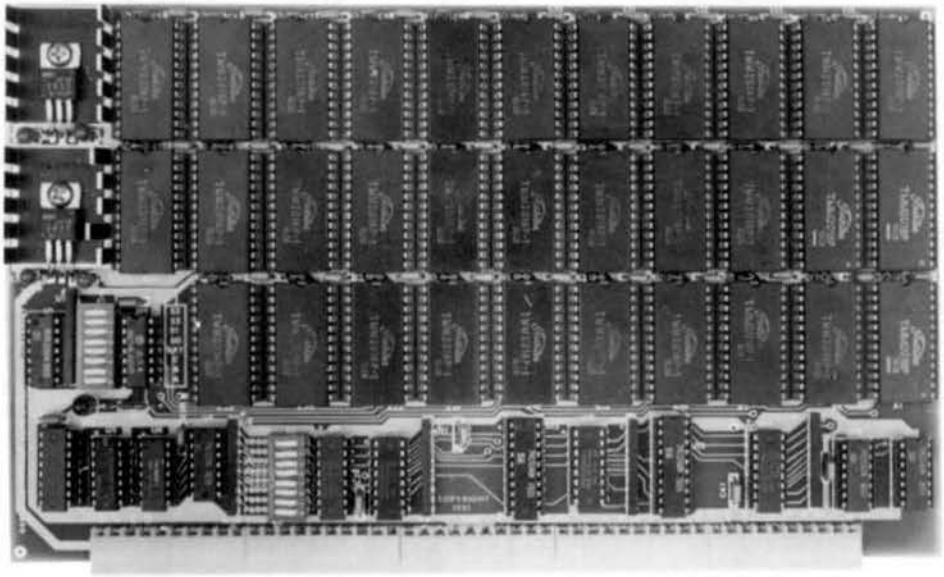
$179.00 (48K KIT)

NEW!

NEW!

LOW POWER!

RAM OR EPROM!



BLANK PC BOARD WITH DOCUMENTATION $52

SUPPORT ICs + CAPS -	$18.00
FULL SOCKET SET -	$15.00

ASSEMBLED AND TESTED **ADD $50**

FEATURES:

- ★ Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- ★ Fully supports Extended Addressing.
- ★ 64K draws only approximately 500 MA.
- ★ 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- ★ Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
- ★ 2716 EPROMs may be installed anywhere on Board.
- ★ Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
- ★ One Board supports both RAM and EPROM.
- ★ RAM supports 2MHZ operation at no extra charge!
- ★ Board may be partially populated in 16K increments.

56K	$219
64K	$249

CLOSE OUT SPECIAL
WE HAVE DROPPED OUR 32K SS-50 STATIC RAM BOARD WHICH USED 2114 LOW POWER RAMS. WE WILL SELL THE REMAINING STOCK OF BLANK PCB'S WITH DATA FOR $17.50 EA. THESE FORMERLY SOLD FOR $50.

16K STATIC RAMS?

The new 2K x 8, 24 PIN, static RAMs are the next generation of high density, high speed, low power, RAMs. Pioneered by such companies as HITACHI and TOSHIBA, and soon to be second sourced by most major U.S. manufacturers, these ultra low power parts, feature 2716 compatible pin out. Thus fully interchangeable ROM/RAM boards are at last a reality, and you get BLINDING speed and LOW power thrown in for virtually nothing.

Digital Research Computers
(OF TEXAS)
P.O. BOX 401565 • GARLAND, TEXAS 75040 • (214) 271-3538

TERMS: Add $2.00 postage. We pay balance. Order under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

AMX

Real-Time Multitasking Executive

for 8080, Z80, 6809 and 8086

Gives your application a head start

Save time and money in the development of your product or system by using AMX, the software executive with proven, fault-free operation.

SIMPLE OPERATION

You divide complex control programs into a number of separate, more manageable programs, called *tasks*, each designed to do one job. This allows tasks to be written and tested separately and then combined to form a reliable, finished system.

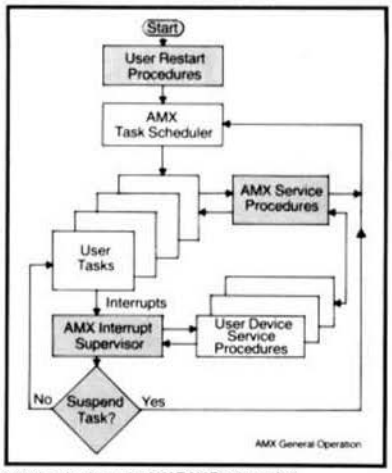
AMX supervises the orderly execution of these tasks, assuring that the most important jobs always get done first. Tasks appear to be executing simultaneously. It's almost like having a separate CPU for each task!

HARDWARE INDEPENDENCE

AMX does not require a particular hardware configuration. *You* control your environment. *You* pick the I/O method. *You* decide the preferred interrupt service technique for your system. AMX will support you on the microprocessor of your choice.

AMX is fast, compact, and ROMable. The AMX nucleus, less than 1400 bytes in size, features multiple task priorities, intertask message passing with priority queuing, external event synchronization, and interval timing.

Support modules provide extended memory management, buffer control and resource allocation. Fast, reentrant integer and floating point math libraries are also available.



AMX is a trademark of KADAK Products Ltd.
CP/M is a trademark of Digital Research Corp.
Z80 is a trademark of Zilog Corp.

AMX interfaces support programs written in C, PASCAL, PL/M, FORTRAN and assembler.

Access to CP/M® disk files in real time is possible using the AMX I/O Supervisor.

COMPLETE DOCUMENTATION

We deliver AMX source on diskette to permit AMX to be moved to the software development system of your choice. Our liberal license agreement permits binary (object) distribution without royalties.

HOW TO ORDER

A specification sheet and price list are available, free. Your check or money order for $75 will purchase the AMX Reference Manual for immediate evaluation (specify 8080, Z80, 8086 or 6809 processor). Add $25 for postage and handling outside USA and Canada. The standard 8080/Z80 AMX Multitasking Executive package, including source code, is **$800.** Support modules and interfaces are available separately.

AMX is the choice of professionals the world over. Make it yours, today.

KADAK Products Ltd.

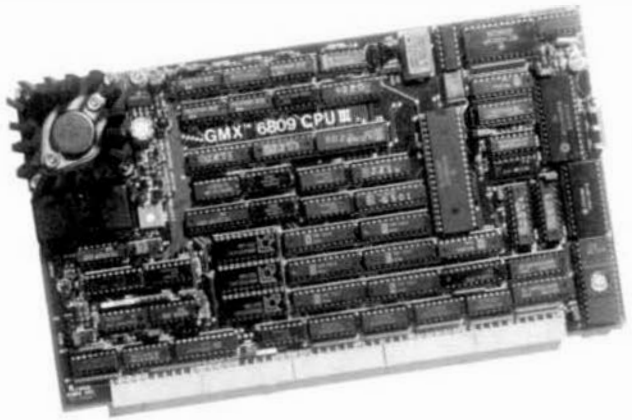
206–1847 W. Broadway Ave., Vancouver, B.C., Canada V6J 1Y5/Phone: (604) 734-2796 Telex: 04-55670

68' MICRO JOURNAL ADVERTISERS INDEX

This index is provided as a reader service. The publisher does not assume any liability for omissions or errors.

U.S. Postal Service Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685): 1. Title of Publication: 68 Micro Journal 2. Publication no: 468510: 2. Date of Filing: 09-28-83: 3. Frequency of Issue: Monthly: A. No. of Issues Published Annually: 12. B Annual Subscription Price: $24.50 4 and 5: Complete Mailing Address of Known Office, Headquarters or General Business Offices of the Publisher: 5900 Cassandra Smith Rd., Hixson, TN. 37343 6. Full Name and Complete Mailing Address of Publisher: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343. Editor: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343., Managing Editor: Larry E. Williams, 5900 Cassandra Smith Rd., Hixson, TN 37343 7. Owner: Computer Publishing Inc., 5900 Cassandra Smith Rd., Hixson, TN 37343, whose Stockholders are: Donald M. Sr., Frances J., Larry E., Mary E., Thomas E. Williams. 8. Known Bondholders, Mortgagees, and other Security Holders Owning or Holding 1 Percent or more of Total Amount of Bonds, Mortgages of other Securities: None 9. For Completion By Nonprofit Organizations Authorized To Mail At Special Rates: N/A 10. Extend and Nature of Circulation: A. Total No. Copies (Net Press Run): Average No. Copies Each Issue During Preceding 12 Months: 10506., Actual No. Copies Of Single Issue Published Nearest To Filing Date: 10400. B. Paid Circulation: 1. Sales Through Dealers And Carriers, Street Vendors And Counter Sales: Average No. Copies Each Issue During Preceding 12 Months: 4631. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 4394. 2. Mail Subscriptions: Average No. Copies Each Issue During Preceding 12 Months: 4881. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 5475. C. Total Paid Circulation: Average No. Copies Each Issue During Preceding 12 Months: 9512. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 9869. D. Free Distribution By Mail, Carrier Or Other Means Including Samples, Complimentary, And Other Free Copies: Average No. Copies Each Issue During Preceding 12 Months: 137. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 146. E. Total Distribution (Sum Of C And D): Average No. Copies Each Issue During Preceding 12 Months: 9649, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 10015. F. Copies Not Distributed: 1. Office Use, Left Over, Unaccounted, Spoiled After Printing: Average No. Copies Each Issue During Preceding 12 Months: 257, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 385. 2. Return From News Agents: Average No. Copies Each Issue During Preceding 12 Months: 600, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 0. G. Total (Sum of E, F1 and 2--Should Equal Net Press Run Shown In A): Average No. Copies Each Issue During Preceding 12 Months: 10506, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 10400. 11. I Certify That The Statements Made By Me Above Are Correct And Complete: (Signed): MARY E. ROBERTSON, Office Manager, 68 Micro Journal. Computer Publishing, Inc. 12. For completion by publishers mailing at the regular rates: Permission requested.

GIMIX STATE OF THE ART 6809 SYSTEMS FOR THE SERIOUS USER.



GIMIX has 19MB or high performance 47MB Winchester Drive Systems and/or Floppy Disk Drive Systems.

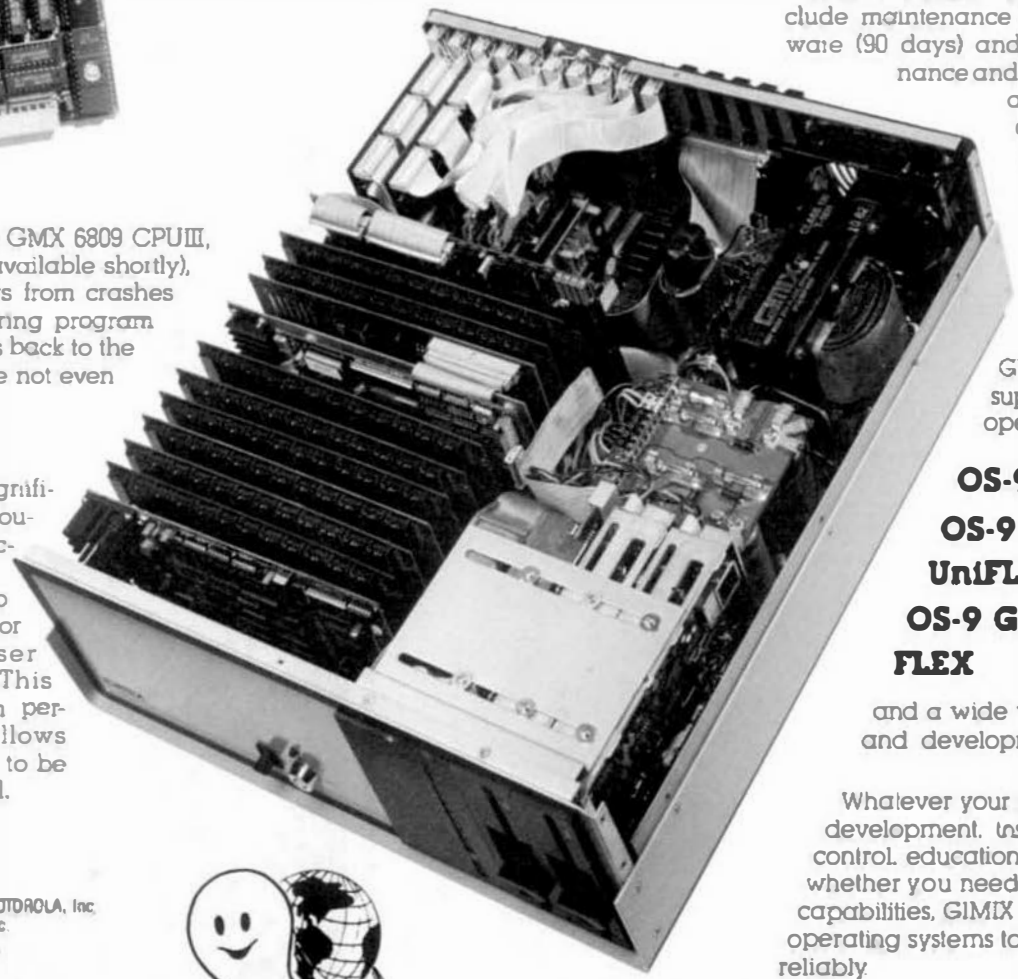
For the user who appreciates the need for a bus structured system using STATIC RAM and powered by a ferro resonant constant voltage transformer.

GIMIX has single user systems that can run both FLEX and OS-9 or Multi user systems for use with UniFLEX or OS-9.

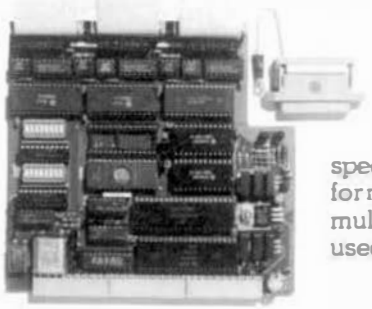
GIMIX versions of OS9 and UniFLEX include maintenance and support by Microware (90 days) and TSC (1 year). Maintenance and support after this period are available at extra cost.

(NOTE: this support and maintenance is only for use with approved GIMIX hardware)

For the ultimate in performance, the Unique GMX 6809 CPUIII, using either OS-9-GMXIII or UniFLEX GMXIII (available shortly), gives protection to the system and other users from crashes caused by defective user programs. e.g. During program development, a programmer who crashes goes back to the shell or the debugger, while the other users are not even aware anything occurred.



The intelligent serial I/O processor boards significantly reduce system overhead by handling routine I/O functions, thereby freeing up the host CPU for running user programs. This speeds up system performance and allows multiple terminals to be used at 19.2K baud.



GIMIX 6809 systems support five predominant operating systems

OS-9 GMX III,
OS-9 GMX II,
UniFLEX,
OS-9 GMX I,
FLEX

and a wide variety of languages and development software.

Whatever your application, software development, instrumentation, process control, educational, scientific or business, whether you need single or multi-user capabilities, GIMIX has hardware and the operating systems to get the job done reliably.

Please phone or write if you need further information.

GIMIX INC.

1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

The Original FLEX™ for Color Computers

- Upgrade to 64K
- RS to FLEX, FLEX to RS file transfer ability
- Create your own character set
- Automatic recognition of single or double density and single or doubled sided
- All features available for either single or multiple drive systems
- Settable Disk Drive Seek Rates
- Faster High Resolution Video Display with 5 different formats
- Save RS Basic from RAM to Disk
- Move RS Basic to RAM
- Load and save function on FLEX disk
- 24 Support Commands 12 with Source Text
- External Terminal Program

DATA-COMP has everything you need to make your TRS-80C Color Computer WORK for YOU! from Parts and Pieces to Full Ready To Use Systems. DATA-COMP designs, sells, services, and SUPPORTS Computer SYSTEMS, not just Software. CALL DATA-COMP TODAY to make your Computer WORK FOR YOU!

System Requirements

FLEX9 Special General Version + Editor & Assembler (which normally sell for $58.00 ea.)	$150.00
F-[illegible](RS) FLEX9 Conversion Rout. for the RS Disk Controller when purchased with Special General FLEX9 Sys.	$ 49.95
when purchased without the General FLEX9 Sys.	$ 59.95
Color Computer with 64K RAM and EXT.BASIC	$399.95

SPECIAL SYSTEM PACKAGES

64K Radio Shack COLOR COMPUTER, Radio Shack COLOR DISK CONTROLLER, a Disk Drive System, Special General Version of FLEX9, F-MATE(RS), and a Box of 10 Double Density Diskettes, a COMPLETE ready to run SYSTEM on your Color TV Set.

$999.95

64K Radio Shack COLOR COMPUTER, Radio Shack COLOR DISK CONTROLLER, a Disk Drive System, and a Box of 10 Double Density Diskettes, a COMPLETE ready to run SYSTEM on your Color TV Set.

$799.95

DISK DRIVE PACKAGES, etc.

These Package include the Radio Shack Disk Controller, Disk Drives with Power Supply and Cabinet, and Disk Drive Cable:

PAK #1 - 1 Single Sided, Double Density Sys.	$389.95
PAK #2 - 2 Single Sided, Double Density Sys.	$675.95
PAK #3 - 1 Double Sided, Double Density Sys.	$569.95
PAK #4 - 2 Double Sided, Double Density Sys.	$919.95
PAK #5 - 2 Qume Thinline Double Sided, Double Density Sys.	$699.95
PAK #6 - 2 Tandon Thinline Single Sided, Double Density Sys.	$599.95

PRINTERS

EPSON RX-80	$375.00
EPSON MX-100	$725.00
EPSON MX-80	$395.00
EPSON FX-80	$599.00

SERIAL BOARDS

MX-Series	$119.95
RX-FX-Series	$ 99.95
Spectrum MX-Series	$ 54.95
Mark Data Keyboards	$ 67.95

FLOPPY DISKETTES

	MEMOREX	VERBATIM
5" Soft Sector Disks		
Single Sided Single Density	$2.40ea.	$2.75ea
Single Sided Double Density	$2.40ea	$2.75ea
Double Sided Double Density		$4.92ea
Plastic Storage Box		$2.00ea
8" Soft Sector Disks		
Single Sided Single Density		$3.75ea
Single Sided Double Density		$4.10ea
Double Sided Double Density		$4.75ea
Plastic Library Box		$5.00ea

PARTS AND PIECES

Radio Shack Disk Controller	$169.95
1 Single Sided, Double Density Disk Drive Tandon	$229.95
1 Double Sided, Double Density Disk Drive Qume	$319.95
1 Qume Thinline Double Sided, Double Density	$249.95
1 Tandon 40 Track SSDD with Cabinet and Power Supply	$259.95
Single Drive Cabinet with Power Supply	$ 79.95
Double Drive Cabinet with Power Supply	$109.95
Single Drive Disk Cable for RS Controller	$ 19.95
Double Drive Disk Cable for RS Controller	$ 24.95
Micro Tech. Prods. Inc. LOWER CASE ROM Adapter	$ 74.95
Radio Shack BASIC Version 1.1 ROM or 1.2 ROM	$ 34.95
Radio Shack Extended Basic ROM	$ 74.95
Radio Shack Disk Basic ROM 1.1	$ 29.95
Screen Clean - Cleans Up Video Distortion On Your Color Computer	$ 19.95
Set of Eight 64K RAM Chips w Mod Instructions	$ 49.95

For Ordering Call TOLL FREE 1-800-338-6800

LAST OF PRODUCTION!!!

!! LIMITED QUANTITY !!

SWTPC 8212 & 8212(W)
Intelligent Terminals

New & Demo Models

At Discount Prices

Remaining Supply of
SWTPC 8212 CRT Terminals

EM-82 Video Terminal

Emulates the 8200 Series from SWTPC

MasterCard VISA

DATA-COMP

P.O. Box 794 *HIXSON, TN 37343*

1-615-842-4601

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

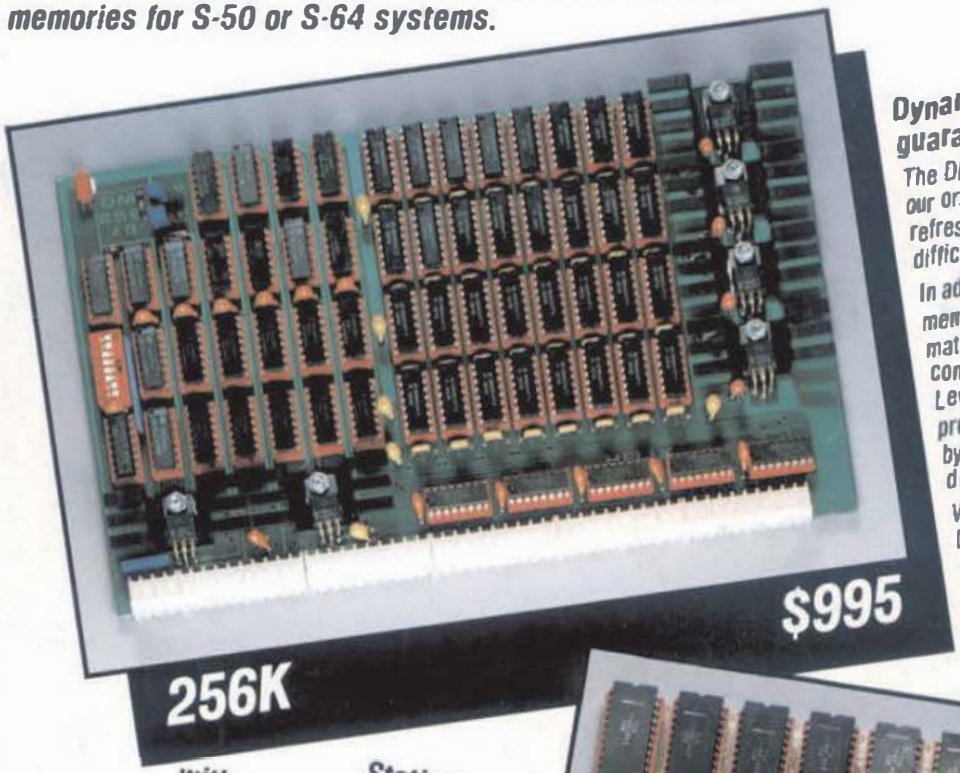
Data-Comp--South East Media & 68 Micro Journal Are Divisions of CPI

68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Second Class Postage Paid
At Chattanooga, TN
ISSN 0194-5025

MEMORY • MEMORY • MEMORY • MEM

Hazelwood Computer Systems presents a pair of advanced memories for S-50 or S-64 systems.



256K **$995**

Dynamic memory with guaranteed 2.5 MHz operation

The DM-256 utilizes the same proven refresh method as our original DM-64. This technique allows fully transparent refresh while maintaining full bus speed and eliminating difficult timing problems.

In addition, individual power regulators for each row of memory ICs enhance reliability by providing the u ti-mate in noise decoupling. Flexible addressing mode controls make the entire 256K available to OS9 Level 2 or UniFLEX. For FLEX, the furnished utility program HYPERDSK,™ formats memory not used by FLEX as a virtual, superfast disk, thus tremendously improving total system performance.

With several hundred already in use, the DM-256 is the best choice for economical, reliable memory.

Assembled, tested, and burned in

Order: DM-256

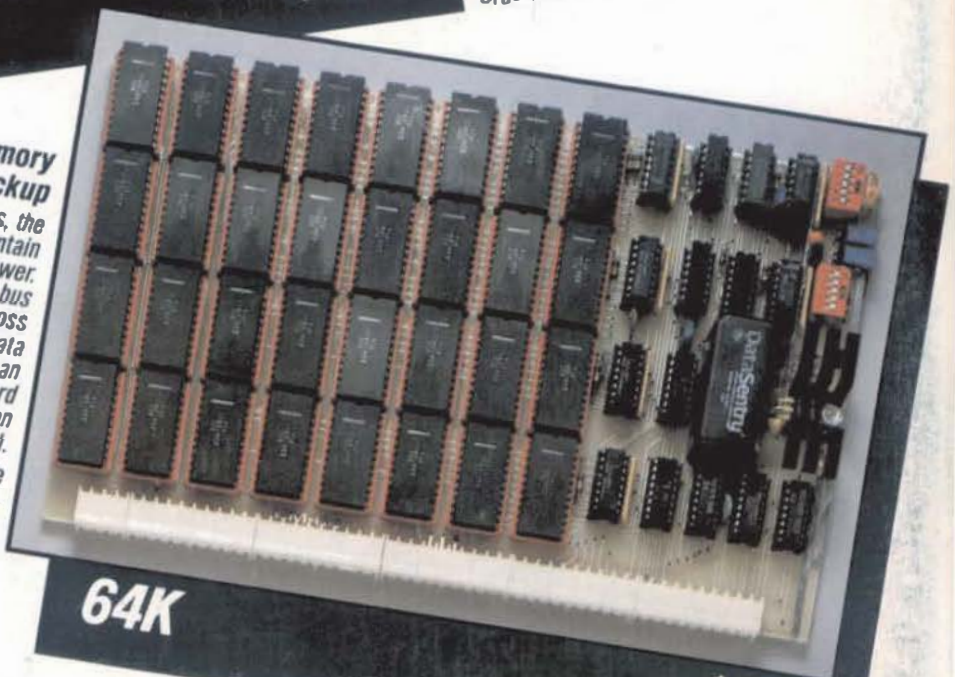
Static memory with on-board battery backup

Utilizing the latest in low power CMOS static RAMs, the CM-64 has an on-board NiCad battery to maintain memory contents in the absence of bus power. Generous timing margins allow operation at bus speeds in excess of 2 MHz. Zener diode power loss sensing inhibits write operations to prevent data loss during power transitions. Write access can be explicitly controlled by means of an on-board jumper, a remote mounted switch, or an external logic signal.

This allows the memory to appear as ROM to the computer after the contents have been "program-med" using normal computer operations. The economical price makes the advantages of non-volatile memory available to any system.

Assembled, tested, and burned in

Order: CM-64



64K **$395**

HAZELWOOD COMPUTER SYSTEMS

907 East Terra, O'Fallon, MO 63366. 314-281-1055

HELIX™

HYPERDSK is a trademark of Hazelwood Computer Systems • OS9 is a trademark of Microware • FLEX and UniFLEX are trademarks of TSC